creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학박사학위논문

기하학적 방법론을 이용한 다양체 표현 학습

Geometric Methods for Manifold Representation Learning

2023 년 2 월

서울대학교 대학원

기계공학부

이 용 현

기하학적 방법론을 이용한 다양체 표현 학습

Geometric Methods for Manifold Representation Learning

지도교수 박 종 우

이 논문을 공학박사 학위논문으로 제출함

2022 년 10 월

서울대학교 대학원

기계공학부

이 용 현

이용현의 공학박사 학위논문을 인준함

2022 년 12 월

위 원 장 : 김 아 영 (인)

부위원장 : 박 종 우 (인)

위 원 : 김 태 섭 (인)

위 원 : 오 민 환 (인)

위 원 : 노 영 균 (인)

Geometric Methods for Manifold Representation Learning

Yonghyeon Lee

Submitted in Partial Fulfillment of The
Requirements for The Degree of

Doctor of Philosophy

in the
Department of Mechanical and Aerospace Engineering

at
Seoul National University

February 2022

ABSTRACT

Geometric Methods for Manifold Representation Learning

by

Yonghyeon Lee

Department of Mechanical and Aerospace Engineering
Seoul National University

Observations from real-world problems are often high-dimensional, and it is very challenging to directly apply machine learning algorithms to data living in such a high-dimensional space, known as the curse of dimensionality. The manifold hypothesis assumes that high-dimensional data lie approximately on some lower-dimensional manifold, suggesting that the data set initially described by many variables can actually be described by a much smaller number of variables. Discovering the lower-dimensional manifold structure and finding its representation – which we call the manifold representation – is one of the most fundamental problems in machine learning.

Autoencoders – which consist of the encoders that map high-dimensional data points to their low-dimensional representations (i.e., latent values) and decoders that map the latent values back to their original data points – are widely used to identify the underlying lower-dimensional manifold and its representation space, simultaneously. In this thesis, we address two fundamental problems of vanilla autoencoders: (i) *the wrong manifold problem* – they often produce manifolds that overfit to noisy training data or have the wrong local connectivity and geometry –, and (ii) *the distorted latent space problem* – they learn geometrically distorted latent representations in the sense

that distances and angles between data points are not well-preserved in the representation space.

Because the manifold is usually not flat but rather a curved space, taking into account the underlying geometry of the manifold is crucial to ensuring good results that do not depend on, e.g., the choice of coordinates. Not surprisingly, existing autoencoder methods for the most part focus on the latent space distributions that are entirely determined by the encoders, yet little if any consideration has been given to the decoders, and they often fail to correctly account for the underlying geometry of the data. This thesis presents a class of autoencoder-based algorithms for manifold representation learning that address these shortcomings. One of the interesting findings in this thesis is that the decoder plays an equally or sometimes more important role than the encoder in autoencoder-based representation learning.

The proposed geometric methods either exploit a priori constructed neighborhood graph or pre-designed Riemannian metric in data space to formulate new loss functions for learning correct manifolds and geometry-preserving latent space coordinates. In particular, in our Riemannian geometric formulations – when we assume and adopt the Riemannian geometry of the data space –, we pay special attention to the coordinate-invariance properties of the regularization terms so that they capture geometrically meaningful quantities. Experiments with a wide range of image, motion capture, and point cloud data sets confirm that, compared to existing state-of-the-art methods, our methods learn the manifold more accurately and with less distortion, improving performance for standard downstream tasks such as image retrieval, clustering, and classification, in some cases by significant margins.

Keywords: Manifold learning, Autoencoder, Riemannian geometry, Isometric representation.

Student Number: 2018-20161

Contents

List of Tables

List of Figures

1 Introduction

1.1 Manifold Hypothesis in Machine Learning

Observations from real-world problems are often high-dimensional, i.e., a large number of variables is used to numerically represent the observed data. For example, consider an RGB image data of size 128×128. The number of variables needed to numerically represent this image is $49,152 = 3 \times 128 \times 128$ (3 is from RGB intensities), meaning that this image data is living in a high-dimensional data space $\mathbb{R}^{49152}$. It is very challenging to analyze data living in such a high-dimensional space, because when the dimensionality increases, the number of data required to fill the the volume of the entire space increases so fast that the available data become inevitably sparse, and thus to obtain a reliable result, the amount of data needed often grows exponentially with the dimensionality, known as the **curse of dimensionality** [1].

Manifold hypothesis, which assumes that high-dimensional data lie approximately on some lower-dimensional manifold, suggests that many data sets that appear to initially require many variables to describe can actually be described by a comparatively small number of variables [2]; below shows some intuitive examples.

Example 1.1.1. Rotating face image manifold. There is a sequence of pictures of a person turning his head from left to right as shown in Figure 1.1, where each image size is $3 \times 728 \times 1280$. These images can be initially viewed as elements of the high-dimensional image data space $\mathbb{R}^{3\times728\times1280}$, however, they clearly do not fill up the entire image space, but rather form a lower-dimensional subspace. We need only one variable to represent the image, i.e., the angle of the person's head, meaning that the set of images can be interpreted as lying on a one-dimensional subspace or a curve as shown in Figure 1.1.

Example 1.1.2. Robot water-pouring motion manifold. Consider a 7-dof robot's discrete-time water-pouring motion data for a fixed cup position as illustrated in Figure 1.2. The data is given as a sequence of 500 robot poses; thus the dimension of

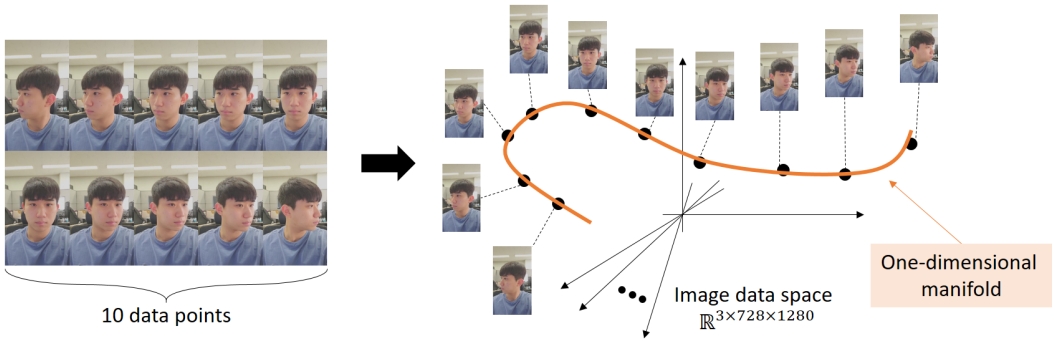


Figure 1.1: A set of face images, where the person turns his head from left to right, approximately lie on a one-dimensional manifold or a curve.

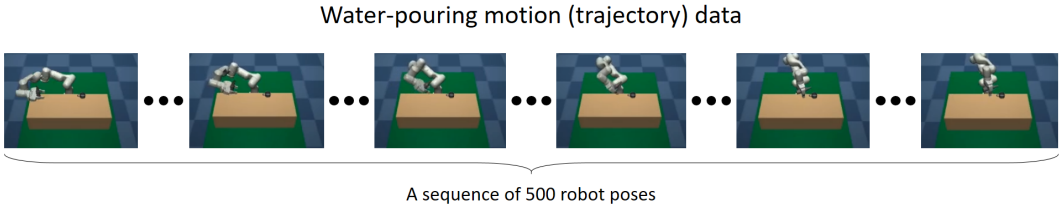


Figure 1.2: An example of a discrete-time robot water-pouring motion data (a sequence of robot poses) for a fixed cup position, where the robot's dof is 7 and the sequence length is 500; thus the data dimension is 7×500.

data is 7×500 which is high-dimensional. Now, suppose we are given 12 water-pouring motion data with varying cup positions as shown in Figure 1.3. These data can be viewed as 12 data points in the high-dimensional robot motion data space $\mathbb{R}^{7\times500}$, however, they clearly do not fill up the entire motion space, but rather form a lower-dimensional subspace. We need two variables to represent the motion, i.e., the xy position of the cup, meaning that the set of motion data can be interpreted as lying on a two-dimensional subspace or a surface as shown in Figure 1.3.

Formally, under the manifold hypothesis, the manifold representation learning problem consists of the following two components (Figure 1.4): (i) identifying the low-dimensional manifold $\mathcal{M}$ of dimension m in the higher-dimensional data space $\mathbb{R}^D$ ($m < D$) and (ii) learning a coordinate system, i.e., a homeomorphism, continuous and invertible map, $f : \mathbb{R}^m \to \mathcal{M} \subset \mathbb{R}^D$ (where we assume $\mathcal{M}$ is homeomorphic to $\mathbb{R}^m$). In the context of machine learning, the coordinate space $\mathbb{R}^m$ is often referred to

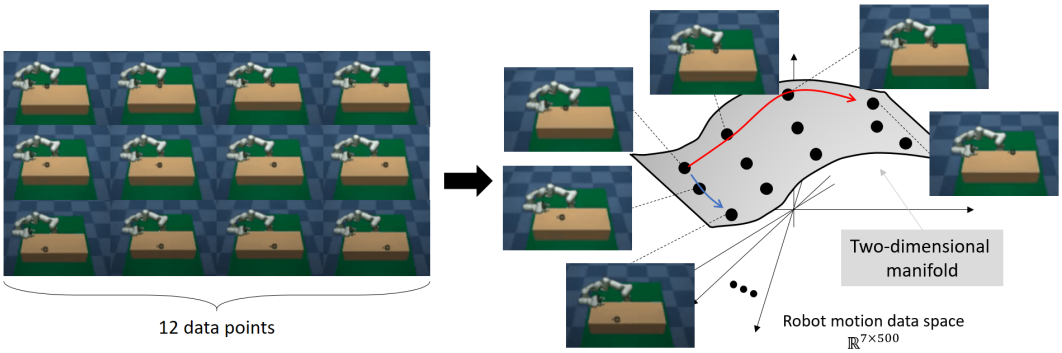


Figure 1.3: A set of water-pouring motion data with varying cup positions approximately lie on a two-dimensional manifold or a surface. Moving in the direction of the red or blue arrow moves the cup position to the right or to the bottom, respectively.

as the latent space or representation space. Then, the coordinate system f and its inverse f^{-1} can be used for making various high-dimensional machine learning problems tractable.

The mapping f can be used to restrict the space of interest in the high-dimensional data space when performing generative or search tasks. For example, to model probability density functions on $\mathcal{M}$, we can push forward density functions on the low-dimensional coordinate space by f. Also, when desired, we can formulate optimal search problems constrained on the learned manifold $\mathcal{M}$ as

$$\min_{x\in\mathcal{M}} \mathcal{J}(x) = \min_{z\in\mathbb{R}^m} \mathcal{J}(f(z)). \tag{1.1.1}$$

The inverse mapping f^{-1} can be used as a feature extractor that maps high-dimensional data to their low-dimensional feature vectors or representations; standard machine learning algorithms (e.g., classification and clustering) can then be applied in the feature space.

1.2 Simultaneous Manifold and Coordinates Learning

Earlier manifold learning approaches mostly focus on finding an embedding of the set of high-dimensional data points to the lower-dimensional space, i.e., $\{x_i \in \mathbb{R}^D\}_{i=1}^N \mapsto \{z_i \in \mathbb{R}^m\}_{i=1}^N$ $(m < D)$, in a way that the local geometrical structures in the graph (the data manifold $\mathcal{M}$ is usually approximated with a neighborhood graph) are preserved in the graph constructed in the lower-dimensional space [3, 4, 5, 6, 7, 8]. These

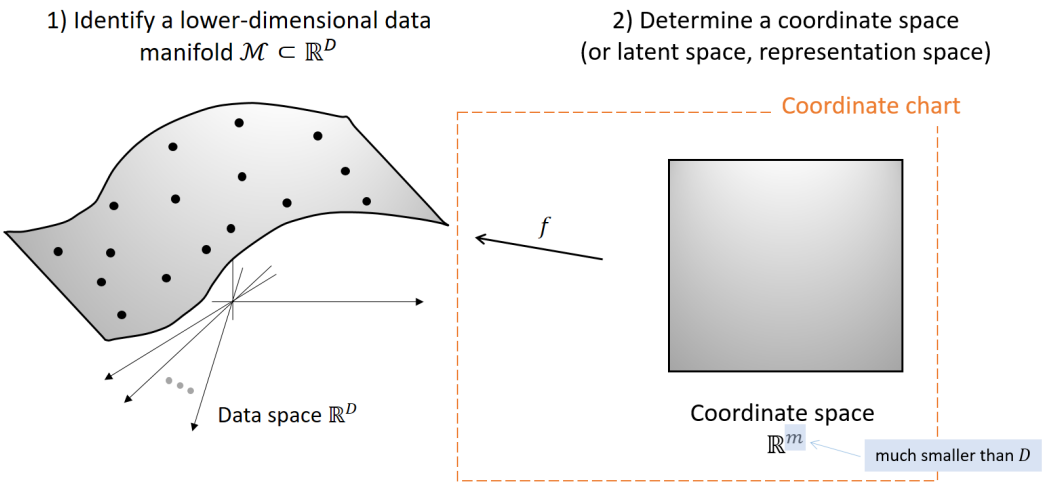


Figure 1.4: Illustration of the two components in the manifold representation learning: (i) identifying the manifold and (ii) determining a coordinate chart.

classical manifold learning methods, however, only find the lower-dimensional representations and do not learn how the manifold actually lies inside the data space $\mathbb{R}^D$. That is, in other words, they can be used for feature extraction but not for data generation.

To know how the manifold is embedded in the ambient data space, we need to learn a mapping $f : \mathbb{R}^m \to \mathbb{R}^D$ so that the image of f corresponds to the manifold of dimension m.[1] In this thesis, we will use the term manifold learning as a synonym for learning this map f. Once the map f that parametrizes the manifold $\mathcal{M}$ is learned, we can find the coordinates of $x \in \mathcal{M}$ in $\mathbb{R}^m$ by solving the inverse problem, i.e., find $z \in \mathbb{R}^m$ such that $x = f(z)$, for example by solving an optimization problem $\min_{z \in \mathbb{R}^m} \|f(z) - x\|^2$. However, solving this coordinates finding problem is computationally very expensive. Hence, it is desired to have a mapping $g : \mathcal{M} \to \mathbb{R}^m$ such that g is a coordinate map that maps a data point on the manifold to its corresponding coordinates.

The main focus of this section is to introduce an effective way of learning the manifold and the coordinate map simultaneously, which is based on the "encoder-decoder" framework often referred to as the **autoencoder** [9]. We will begin by introducing a simple classic example of the linear encoder-decoder. Let $E \in \mathbb{R}^{m \times D}$ be an encoder matrix and $D \in \mathbb{R}^{D \times m}$ be a decoder matrix. A data point $x \in \mathbb{R}^D$ is mapped to

[1] Rigorously, the Jacobian of f should be a full rank everywhere for the image of f to be a manifold.

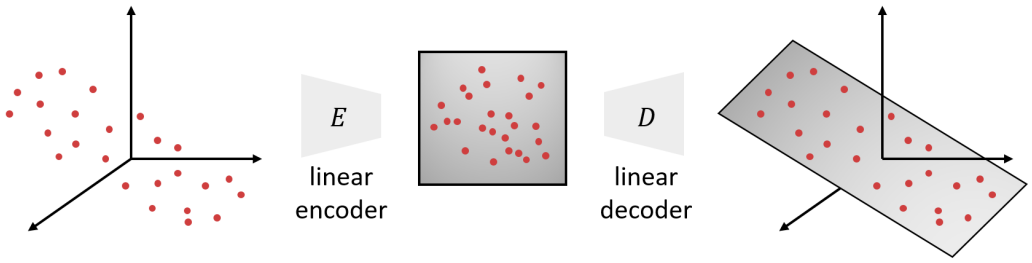


Figure 1.5: Illustration of the linear autoencoder. The linear decoder maps the low-dimensional space to the linear subspace in the higher-dimensional space, and the encoder maps the data points to their coordinates.

the low-dimensional space by the encoder as $z = Ex \in \mathbb{R}^m$ and a point $z \in \mathbb{R}^m$ is mapped back to the data space by the decoder as $x = Dz \in \mathbb{R}^D$. Given a set of high-dimensional data points $\{x_i \in \mathbb{R}^D\}_{i=1}^N$ approximately lying on some lower-dimensional linear subspace, the linear encoder and decoder can be trained simultaneously, so that the decoder discovers the linear subspace and the encoder maps the data points to their coordinates as illustrated in Figure 1.5, with the following reconstruction error minimization framework:

$$\min_{E,D} \sum_{i=1}^{N} \|x_i - DEx_i\|^2. \tag{1.2.2}$$

By minimizing this objective function, the data points become to lie on the image of the decoder D and the encoder E maps x to $z = Ex$ such that z is the solution of the aforementioned inverse problem, i.e., $\min_z \|x - Dz\|^2$. It is well-known that the subspace discovered by the linear decoder D is equivalent to the subspace obtained by the PCA algorithm [10].

While the linear autoencoder can only discover linear manifolds, this framework has been successfully generalized to learn arbitrarily curved manifolds, together with the recent advances in deep learning techniques used for approximating arbitrary complex functions [11]. Similar to the linear case, the core idea is to learn two mappings an encoder $g : \mathbb{R}^D \to \mathbb{R}^m$ and a decoder $f : \mathbb{R}^m \to \mathbb{R}^D$ approximated with deep neural networks so that the composition of them reconstructs the given data points $x_i \in \mathbb{R}^D$, i.e., $f \circ g(x_i) \approx x_i$, for $i = 1, \cdots, N$. One of the most straightforward ways to learn these mappings is to solve the following reconstruction error minimization problem with parametric models g_ϕ and f_θ, each of which is parametrized by ϕ and θ (e.g.,

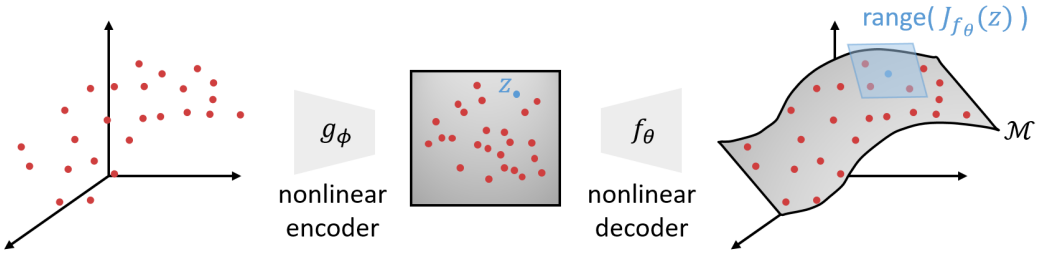


Figure 1.6: Illustration of the nonlinear autoencoder. The decoder maps the low-dimensional space to the nonlinear manifold in the higher-dimensional space, and the encoder maps the data points to their coordinates. In particular, the range of the decoder's Jacobian at z corresponds to the tangent space of the learned manifold $\mathcal{M}$ attached at $f_\theta(z)$.

neural networks):

$$\min_{\theta,\phi} \sum_{i=1}^{N} \|x_i - f_\theta(g_\phi(x_i))\|^2. \tag{1.2.3}$$

Then, the data points lie on the image of the decoder f_θ, and the encoder g_ϕ maps the data points to their coordinates. In other words, the image of the decoder f_θ is the learned manifold $\mathcal{M}$ embedded in $\mathbb{R}^D$ as illustrated in Figure 1.6.

Of particular relevance to this thesis is the geometric property of the decoder's higher-order derivatives, which later gives us the key insight to solving the research problems that we pose in the next section. The first-order derivative or Jacobian denoted by $J_{f_\theta} = \frac{\partial f_\theta}{\partial z}$ at z corresponds to the tangent space of the learned manifold $\mathcal{M}$ attached at $f_\theta(z)$. The m column vectors in the Jacobian matrix $J_{f_\theta}(z) \in \mathbb{R}^{D\times m}$ can be viewed as a set of linearly independent basis vectors for the tangent space attached at $f_\theta(z)$ denoted by $T_{f_\theta(z)}\mathcal{M}$ (under the condition that $J_{f_\theta}(z)$ is full rank). And the second-order derivative, which computes how the Jacobian changes locally, has information about the manifold's curvature. These notions will come important to understanding geometric ideas introduced later.

In concluding this section, we introduce some notations and terminologies used throughout this thesis. The coordinate space $\mathbb{R}^m$ for the learned manifold $\mathcal{M}$ will also be referred to as the latent space and representation space, and its coordinates as the latent coordinates and latent (space) representation. The image of the decoder will be considered as the learned manifold $\mathcal{M}$, which sometimes will be referred to as the decoded manifold, and the decoder f_θ will be treated as the coordinate system. Depending on the context, we will introduce other notations as needed

1.3 Motivation and Research Problems

In vanilla autoencoders, there are two fundamental issues that we aim to address in this thesis: (i) they often produce manifolds that overfit to noisy training data or have the wrong local connectivity and geometry, and (ii) they learn geometrically distorted latent representations where geometric quantities such as the lengths, angles, and volumes are not preserved compared to those in the data manifold. Below, we will take a closer look at the following issues one by one.

1.3.1 Wrong Manifold

In the traditional autoencoder training setting, we are given a set of finite data points $\{x_i \in \mathbb{R}^D\}_{i=1}^N$. Assuming the data points are clean and perfectly lie on the ground-truth data manifold $\mathcal{M}$, the primary condition for a correct coordinate system $f : \mathbb{R}^m \to \mathbb{R}^D$ to satisfy is the following:

$$x_i \in \mathcal{M} \subset \mathbb{R}^D \text{ for all } i \iff x_i \in \text{image}(f) \text{ for all } i. \tag{1.3.4}$$

However, finding such a map f is fundamentally ill-posed, meaning that there are infinitely many mappings f that satisfy the above condition. For example, as shown in Figure 1.7, assume that there exists a ground-truth one-dimensional manifold embedded in the two-dimensional data space and we are given a set of finite two-dimensional data points (blue points). There are many manifolds (orange manifolds) where the given data points perfectly lie; thus, we cannot specify the solution manifold without further assumptions on the manifold or decoder f.

Vanilla autoencoders learn arbitrary manifolds which often overfit the training data or have the wrong local connectivity and geometry; some examples are given below.

Example 1.3.1. CIFAR-10 image data. The CIFAR-10 dataset consists of 32×32 color images in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). A vanilla autoencoder trained with 10000 images (with 128-dimensional latent space) learns the manifold that overfits the training data as illustrated in Figure 1.8; for the (unseen) test image data, the reconstructed images are blurry and very different from the original input images.

Example 1.3.2. Rotated MNIST image data of digit 3. Given a set of rotated MNIST images of digit 3 shown in Figure 1.9 (upper-right), a vanilla autoencoder is trained with one-dimensional latent space. We generate a sequence of digit 3 images by traveling the one-dimensional latent space as shown in the lower-right part of the

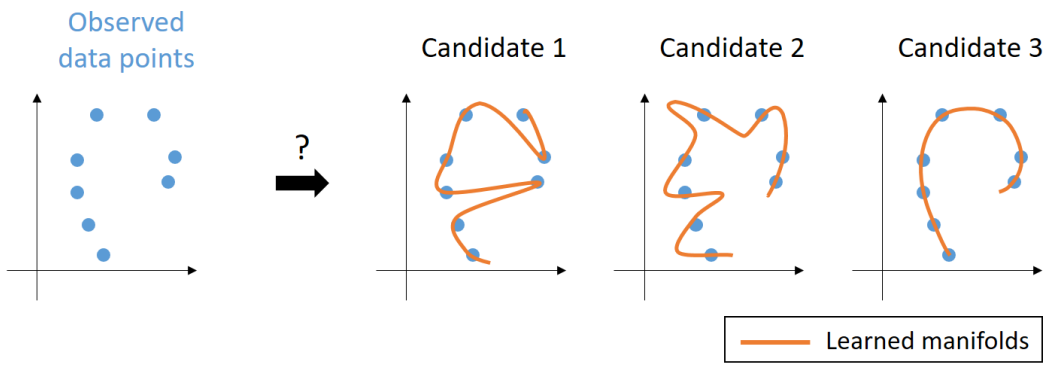


Figure 1.7: Illustration of the fundamental ill-posedness of the manifold learning problem. Given blue data points, there are many manifolds where the given data points perfectly lie. We cannot specify the solution without further assumptions.

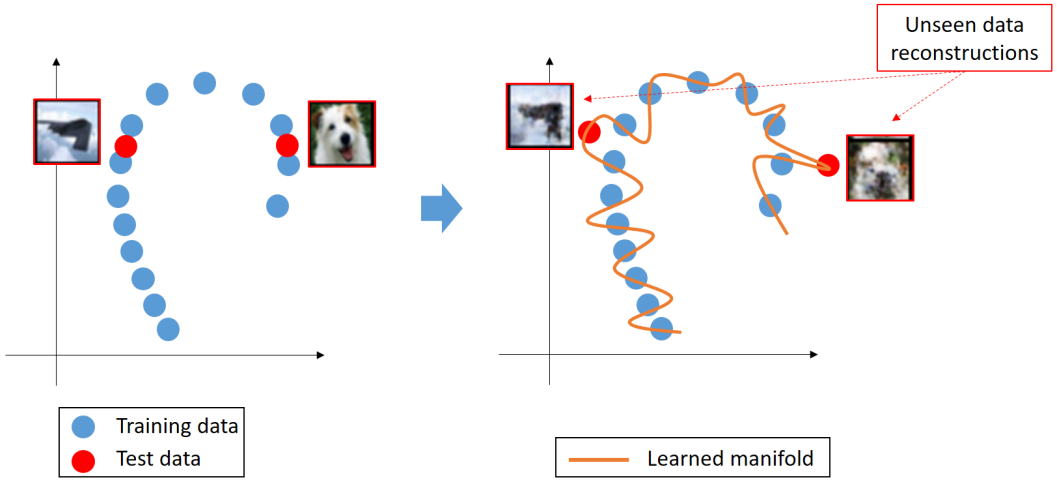


Figure 1.8: A pictorial description of how the learned CIFAR-10 image manifold that overfits the training data (blue points) might look like, and some reconstruction results of the test data (red points). The manifold is indeed a multi-dimensional surface; here it is described as an orange curve just for visualization.

figure. In the generated images, the digit 3 image rotates smoothly but at one point abruptly changes as marked in the orange box, because, as shown in the left part of the figure, the manifold has learned the wrong local connectivity.

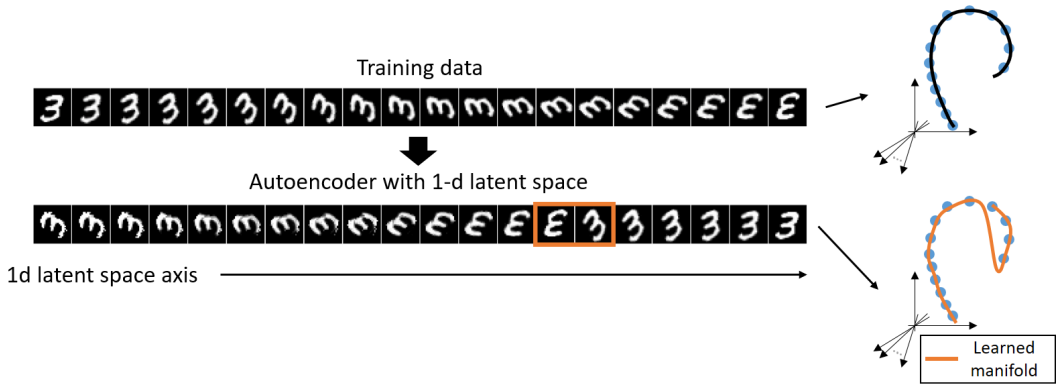


Figure 1.9: *Upper-left:* A set of rotated MNIST images of digit 3 (training data). *Lower-left:* A sequence of digit 3 images generated by traveling the one-dimensional latent space. *Right:* A pictorial description of how the learned one-dimensional manifold (a curve) with the wrong local connectivity might look.

1.3.2 Distorted Latent Space

Given a ground-truth data manifold $\mathcal{M}$, the problem of finding a coordinate system for $\mathcal{M}$ is again fundamentally ill-posed because there exist infinitely many coordinate systems. For example, if $f : \mathbb{R}^m \to \mathcal{M}$ is one coordinate system, then for any continuous and invertible map $h : \mathbb{R}^m \to \mathbb{R}^m$ the composition map $f \circ h^{-1} : \mathbb{R}^m \to \mathcal{M}$ is also another coordinate system (Figure 1.10). Recall the reconstruction error objective function in the vanilla autoencoder $\sum_i \|x_i - f \circ g(x_i)\|^2$, if f^* and g^* are solutions, then for any continuous and invertible map $h : \mathbb{R}^m \to \mathbb{R}^m$ the composition maps $f^* \circ h^{-1}$ and $h \circ g^*$ are also solutions. Therefore, we cannot specify one coordinate system over the others without further assumptions on the mappings.

This is closely related to the classical map-making problem, i.e., making a map of the earth, by finding a mapping from the surface of the earth approximated by a two-dimensional sphere S^2 to a two-dimensional Cartesian plane $\mathbb{R}^2$ (Figure 1.11). As shown in Figure 1.12, a wide variety of maps of the earth exist, each of which is based on different projection methods. In general, maps that are arbitrarily distorted are not preferred, rather it is designed to preserve specific intrinsic geometric quantities defined based on the purpose of the map. For example, the Mercator projection preserves angles while the areas are distorted, and the Gall-Peters map preserves areas although the shapes are distorted (Figure 1.13).

Vanilla autoencoders often learn arbitrarily distorted coordinate spaces (e.g., coordinate space 1 in Figure 1.10) – by being distorted, we mean that straight lines on the

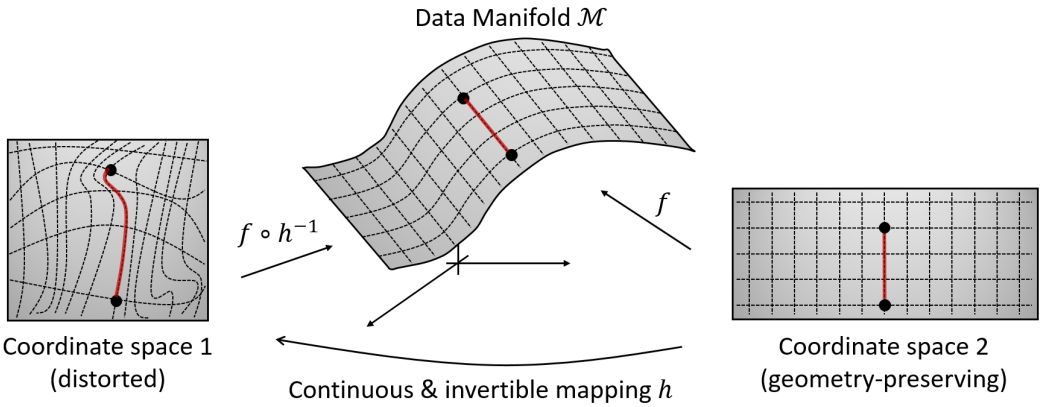


Figure 1.10: There are many coordinate systems for a data manifold $\mathcal{M}$; some produce geometrically distorted coordinate spaces.

coordinate space do not correspond to the actual shortest paths (i.e., geodesics) on the manifold – and standard machine learning algorithms applied to their representations often show less-than-desirable performances on downstream tasks; some examples are provided below.

Example 1.3.3. Interpolation. Let $f : \mathbb{R}^m \to \mathcal{M}$ be a decoder or coordinate system for the learned manifold $\mathcal{M}$, and $g : \mathcal{M} \to \mathbb{R}^m$ be its encoder or approximate inverse of f, i.e., $x \approx f(g(x))$ for all $x \in \mathcal{M}$. Given two data points $x_1, x_2 \in \mathcal{M}$, one of the simplest interpolation methods is to linearly interpolate their latent representations and decode them back to the data manifold, i.e., $f(g(x_1) + (g(x_2) - g(x_1)) \times t) \in \mathcal{M}$ for $t \in [0, 1]$. However, when the coordinate space is distorted, the latent space linear paths do not produce the shortest paths (i.e., geodesics) on the manifold, and thus they may not produce smooth interpolation results along the data manifold.

Figure 1.14 shows some latent space linear interpolation examples compared to actual geodesic paths on the learned manifold. We provide three examples: (a) MNIST image manifold of digits 3, 6, 8, 9 produced by autoencoder with two-dimensional latent space, (b) point cloud manifold of shapes cylinder, ellipsoid, cone produced by autoencoder with two-dimensional latent space, and (c) human pose manifold from motions walking, balancing, punching, jogging produced by autoencoder with eight-dimensional latent space. Linear interpolants are shown in upper rows while the geodesic interpolants are shown in lower rows, and abrupt changes are marked by the orange boxes. Overall, sudden changes occur in the middle of the linear interpolants while

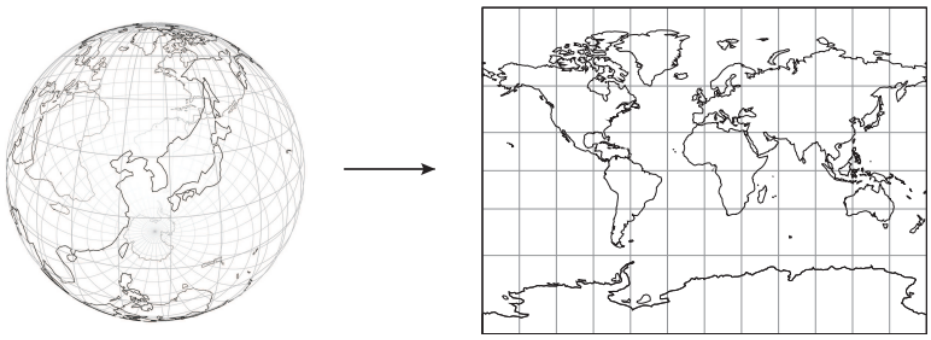


Figure 1.11: Map-making: finding a mapping from a sphere to a Cartesian plane, adopted from Figure 1.5 in [12].

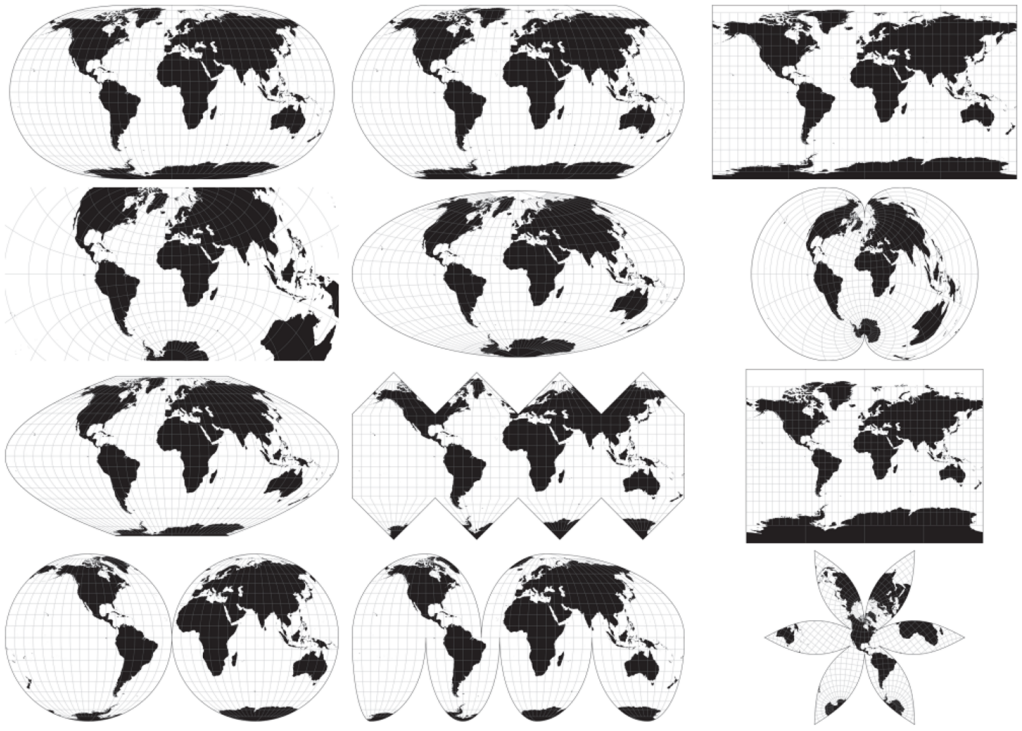
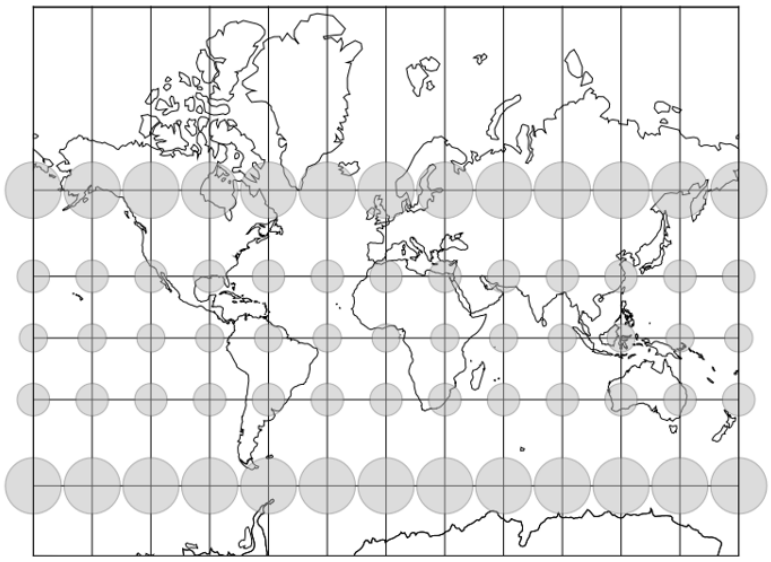
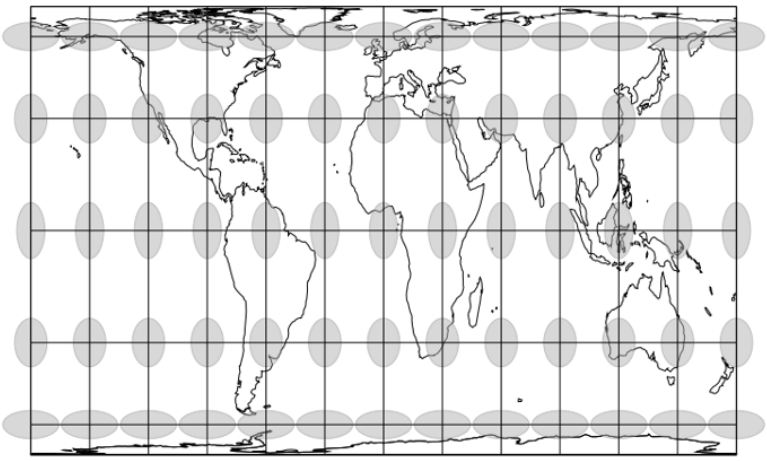


Figure 1.12: Maps of the earth, adopted from Figure 1.6 in [12].



The Mercator projection (angle-preserving)



The Gall-Peters projection (area-preserving)

Figure 1.13: Two examples of maps of the earth, adopted from Figure 1.7 in [12].

geodesic interpolants are smooth, which shows that autoencoders have learned distorted latent representations.

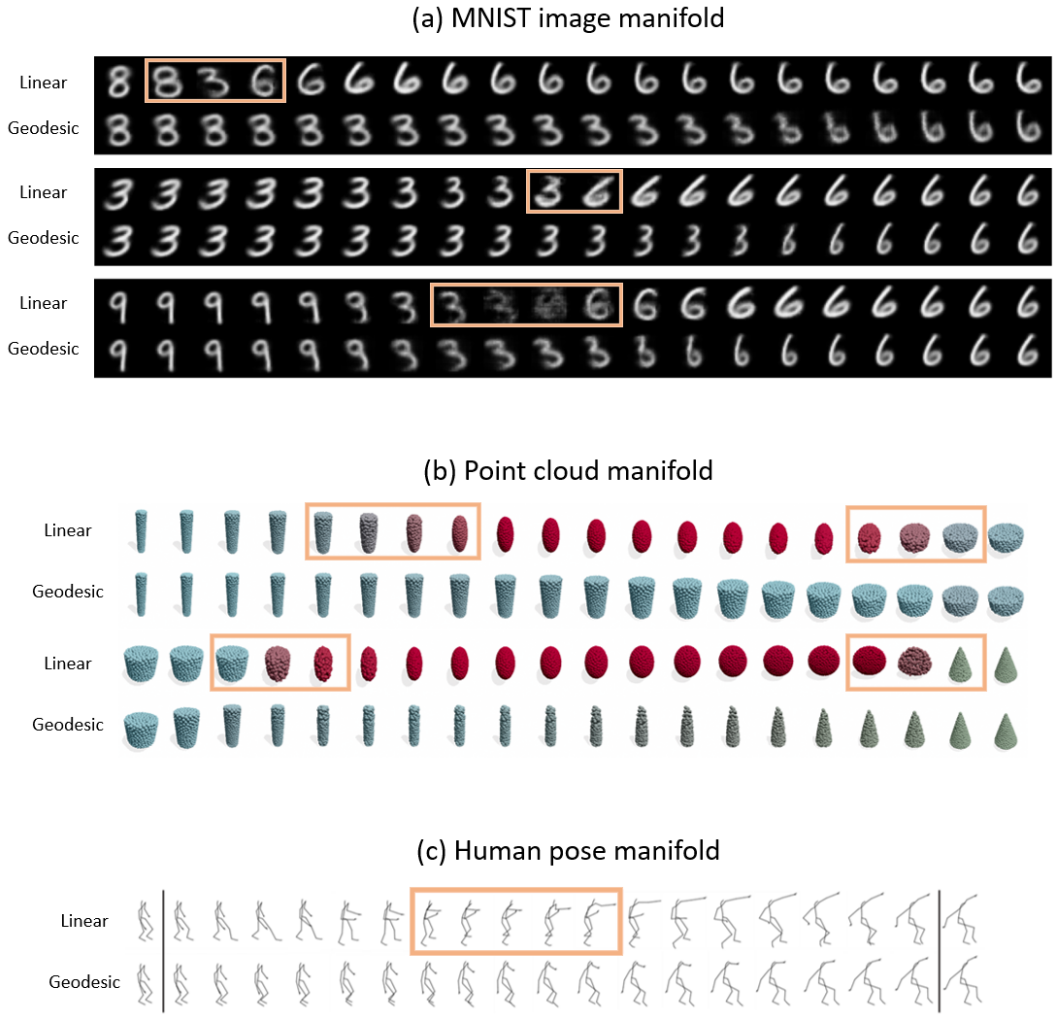


Figure 1.14: Interpolation examples on various manifolds train with autoencoders: (a) MNIST image manifold of digits 3, 6, 8, 9, (b) point cloud manifold of shapes cylinder, ellipsoid, cone, and (c) human pose manifold from motions walking, balancing, punching, jogging. *Upper:* Linear interpolants in the distorted latent space. *Lower:* Geodesic interpolants along the learned manifold. Abrupt changes are marked by orange boxes.

Example 1.3.4. Clustering. The goal of clustering is to divide data points into groups so that data points in the same group are more similar to other data points in the same group than data points in other groups. While directly applying standard clustering algorithms to the high-dimensional data space shows poor performances (recall the curse of dimensionality), autoencoder can produce a lower-dimensional representation space where the algorithms can be instead applied. In general, clustering in the low-dimensional latent space produces much better results than that in the original high-dimensional data space.

However, when the latent space is distorted, latent space clustering algorithms still do not produce satisfying results. In such cases, the Euclidean distances do not capture the actual data distances along the data manifold (i.e., geodesic distances), and thus, although the cluster structure is clear in the data manifold, it may not be in the distorted latent space.

Figure 1.15 shows some latent space clustering results by using the Gaussian mixture model. We provide two examples: *Left:* FMNIST image data of classes trouser, sandals produced by autoencoder with two-dimensional latent space, and *Right:* point cloud data of shapes box, cone, ellipsoid produced by autoencoder with two-dimensional latent space. Overall, their clustering structures are not clear in the distorted latent spaces; the Gaussian mixture models cannot separate different classes as can be seen from the samples from each mixture component.

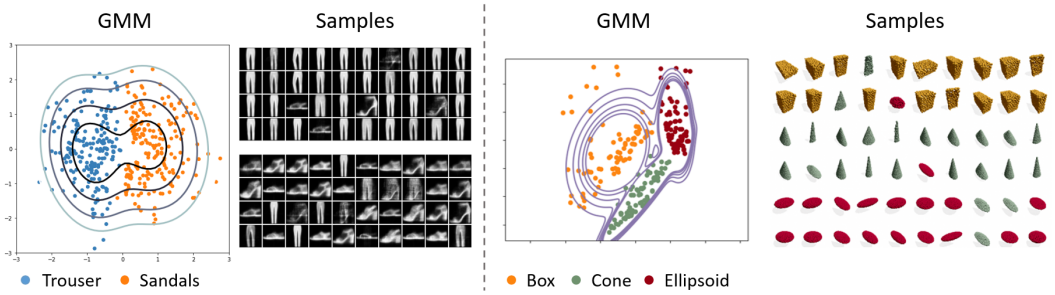


Figure 1.15: Clustering examples on the distorted latent spaces obtained by autoencoders: *Left:* FMNIST image data of classes trouser, sandals, *Right:* point cloud data of shapes box, cone, ellipsoid. Gaussian mixture models fitted to the distorted latent representations cannot find separate clusters.

Example 1.3.5. Optimization on the data manifold. Many machine learning problems can be solved in the following two-step approach: (i) learning a low-dimensional

manifold $\mathcal{M}$ via a mapping from a low-dimensional latent space $\mathbb{R}^m$ to the high-dimensional data space $\mathbb{R}^D$ denoted by $f : z \in \mathbb{R}^m \mapsto x \in \mathcal{M} \subset \mathbb{R}^D$ (called a coordinate system) and (ii) solving an optimization problem with an objective function $\mathcal{J} : \mathbb{R}^{\mathcal{D}} \to \mathbb{R}$ constrained on the manifold as follows:

$$\min_{x \in \mathcal{M}} \mathcal{J}(x) = \min_{z \in \mathbb{R}^m} \mathcal{J}(f(z)). \tag{1.3.5}$$

To solve this type of latent space optimization problem, many existing works use the standard Euclidean optimization algorithms, assuming the Euclidean geometry of the latent space. However, the latent space is in general geometrically distorted so the Euclidean metric in the latent space does not capture the actual geometry of the data space. Therefore, applying the standard Euclidean optimization algorithms in the distorted latent space can be sometimes very inefficient.

Consider an example shown in Figure 1.16, where, given three data points (black dots) on the sphere manifold, we find the intrinsic mean of them in the spherical coordinates (θ^*, ϕ^*) by solving an optimization problem, i.e., the minimization of the sum of the geodesic distances from (θ^*, ϕ^*) to the given points. When the Euclidean gradient descent algorithm is used in the spherical coordinates, because of the geometric distortion between the coordinates and the sphere manifold, the gradient flow (blue) is unnaturally curved toward the north pole and the objective function slowly decreases.

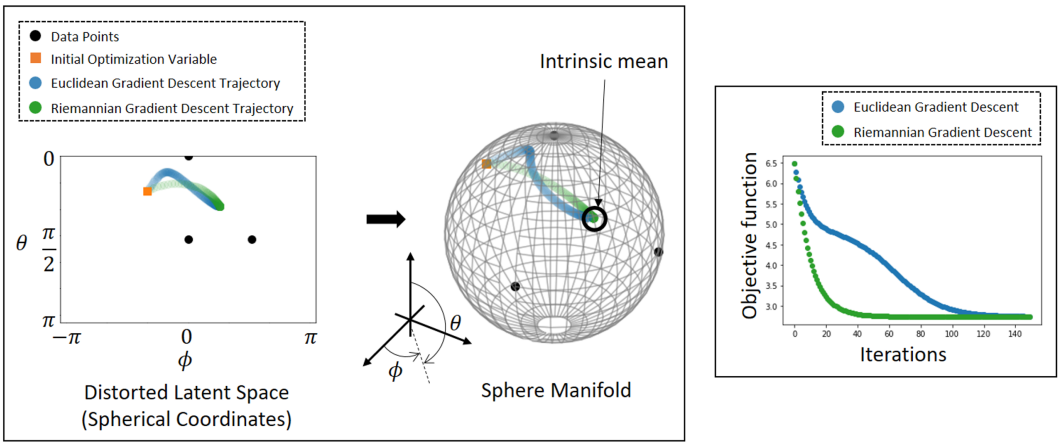


Figure 1.16: *Left*: Euclidean optimization in the distorted latent space produces unnatural gradient flow (blue) on the sphere manifold, whereas the Riemannian gradient flow (green) is smoother and more natural. *Right*: The objective much more quickly decreases with the Riemannian gradient descent method.

1.4 Statistical Interpretations *

In this section, we go beyond the intuitive perspective of autoencoder-based manifold learning and discuss what it means from a statistical perspective in a more formal way. Starting with the simplest case where the data manifold has a constant dimension (with boundary), we prove how the autoencoder can be used to estimate the ground truth manifold. Then we discuss how we should approach the manifold learning problem when the lower-dimensional structure is much more complex – e.g., not a constant-dimensional manifold nor homeomorphic to the Cartesian space – so a single autoencoder cannot represent the manifold explicitly, and how we should measure the manifold learning accuracy in practice. For those who are not familiar with the notions of manifold and its explicit/implicit parametrizations, we recommend to read Chapter 2.1.1 before this section.

Given a high-dimensional data space $\mathbb{R}^D$, suppose there is a probability density function $p_{\text{data}}(x)$ of our interest. We interpret the manifold hypothesis as when the support of $p_{\text{data}}(x)$ has a lower-dimensional structure. To be called a manifold, mathematically, the support should be locally homeomorphic to the Euclidean space at all points, but in this thesis, as the abuse of terminology, we use the data manifold to indicate more complex lower-dimensional structure embedded in the high-dimensional data space although it does not have a constant dimension globally, e.g., a union of manifolds of different dimensions.

To begin with the simplest case, assume that the lower-dimensional structure is a differentiable manifold of constant dimension m with boundary that is homeomorphic to some subset in $\mathbb{R}^m$; denote the manifold by $\mathcal{M} := \text{supp}(p_{\text{data}}(x)) \subset \mathbb{R}^D$. Let $\hat{g} : \mathbb{R}^D \to \mathbb{R}^m$ be an estimate of the encoder and $\hat{f} : \mathbb{R}^m \to \mathbb{R}^D$ be an estimate of the decoder that are trained by some autoencoder training method given a set of training data $\mathcal{D}_{\text{train}} := \{x_i \in \mathbb{R}^D\}_{i=1}^N$ sampled from p_{data}. The autoencoder is a **manifold estimator** in the following sense: (i) let $\hat{U} := \cup_{i=1}^N B_{r_i}(\hat{g}(x_i)) \subset \mathbb{R}^m$ describe the data region in the latent space where r_i is the average of the distances from $\hat{g}(x_i)$ to its k-nearest neighbor points in the latent space ($B_r(z)$ denotes an open ball of radius r centered at z) as shown in Figure 1.17(a) and (ii) the decoding of $\hat{U}$, i.e., $\hat{f}(\hat{U})$, is an estimate of the manifold which we denote by $\hat{\mathcal{M}}(\mathcal{D}_{\text{train}})$ as shown in Figure 1.17(b). There may be other ways to define $\hat{U}$, but for ease of discussion, we will consider it as a union of balls.

The vanilla autoencoder trained to reconstruct the training data

$$\hat{f}, \hat{g} := \min_{f,g \in \mathcal{F},\mathcal{G}} \sum_{i=1}^{N} \|x_i - f \circ g(x_i)\|^2 \tag{1.4.6}$$

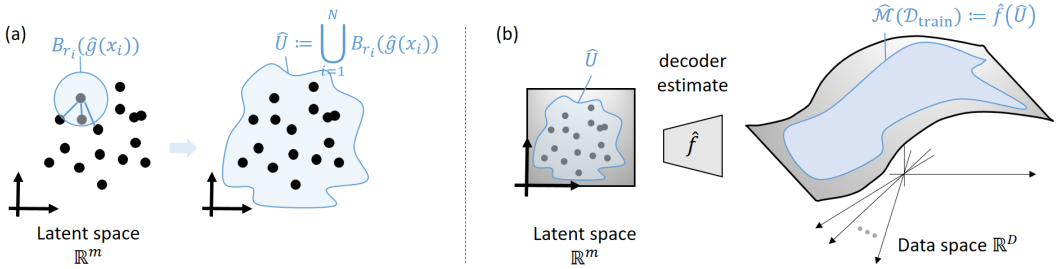


Figure 1.17: (a) The union of balls $\hat{U}$, a subset of the latent space, describes the data region (where the radius of each ball r_i is set to be the average of the 3 nearest neighbor distances) and (b) the manifold is estimated by decoding $\hat{U}$.

with sufficiently expressive model search space $\mathcal{F}, \mathcal{G}$ – so that, as the number of training data increases to infinite, $\hat{f}, \hat{g}$ converge to satisfy $\hat{f} \circ \hat{g}(\mathcal{D}_{\text{train}}) = \mathcal{D}_{\text{train}}$ – is an **asymptotically unbiased manifold estimator**, i.e., $\hat{\mathcal{M}}(\mathcal{D}_{\text{train}}) \to \mathcal{M}$ as $|\mathcal{D}_{\text{train}}| \to \infty$. To see why, consider $\hat{U} = \cup_{z \in \hat{g}(\mathcal{D}_{\text{train}})} B_{r(z)}(z)$; as $|\mathcal{D}_{\text{train}}| \to \infty$, since $r(z) \to 0$, $B_{r(z)}(z) \to z$ and $\hat{U} \to \hat{g}(\mathcal{D}_{\text{train}})$. And $\hat{\mathcal{M}}(\mathcal{D}_{\text{train}}) = \hat{f} \circ \hat{g}(\mathcal{D}_{\text{train}}) \to \hat{f} \circ \hat{g}(\mathcal{M}) = \mathcal{M}$. However, in non-asymptotic situations where the number of training data is limited, the estimated manifolds can be wrong as discussed in the above chapters, and proper regularizations are necessary to learn accurate manifolds.

Meanwhile, the assumption that $\text{supp}(p_{\text{data}})$ is a differentiable manifold of constant dimension homeomorphic to some subset in $\mathbb{R}^m$ in practice mostly does not hold, and rather it is a union of multiple manifolds each of which has different dimensions and topologies. In such cases, a single decoder cannot explicitly parametrize the manifold; to see why, we refer to Figure 2.5. As in many other currently existing autoencoder studies, we give up to satisfy $\hat{\mathcal{M}}(\mathcal{D}_{\text{train}}) = \mathcal{M}$, yet focus on learning the manifold such that $\mathcal{M} \subset \hat{\mathcal{M}}(\mathcal{D}_{\text{train}})$ by choosing a big enough latent space dimension. It is desirable to find the smallest latent space dimension but big enough to implicitly parametrize the lower-dimensional structure (e.g., multiple manifolds) that we call the data manifold with the abuse of terminology.

Satisfying $\mathcal{M} \subset \hat{\mathcal{M}}(\mathcal{D}_{\text{train}})$ is more important than the opposite direction in the manifold representation learning, since it is of utmost importance that representations do not lose information about the data as they reduce dimensionality. Of course, not every point in the union of balls $\hat{U}$ is decoded to data in $\mathcal{M}$, and to address this issue, we need to identify a lower-dimensional data structure lying in the latent space again, which is out of the scope of the thesis.

For the above reasons, in this thesis, we evaluate the manifold learning accuracy

of autoencoder methods by investigating whether $\mathcal{M} \subset \hat{\mathcal{M}}(\mathcal{D}_{\text{train}})$. In practice, we are given a set of test data $\mathcal{D}_{\text{test}} := \{x_i \in \mathcal{M}\}_{i=1}^{M}$ not used for estimating $\hat{f}, \hat{g}$. The most straightforward way to measure how well $\mathcal{D}_{\text{test}}$ lies on $\hat{M}(\mathcal{D}_{\text{train}})$ would be to compute the following error metric:

$$\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{x \in \mathcal{D}_{\text{test}}} \min_{z \in \hat{U}} \|\hat{f}(z) - x\|^2, \tag{1.4.7}$$

where $\hat{U} := \cup_{i=1}^{N} B_{r_i}(\hat{g}(x_i)) \subset \mathbb{R}^m$. However, this metric requires to solve the optimization problem multiple-times, which is computationally expensive. Instead, to reduce the cost, we assume the estimated encoder $\hat{g}$ is good enough to provide an approximate solution to the above optimization, i.e., $\hat{g}(x) \approx \arg\min_{z \in \hat{U}} \|\hat{f}(z) - x\|^2$ for $x \in \mathcal{M}$. Consequently, we use the test set reconstruction error to measure the manifold learning accuracy:

$$\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{x \in \mathcal{D}_{\text{test}}} \|\hat{f} \circ \hat{g}(x) - x\|^2. \tag{1.4.8}$$

1.5 Outline & Contributions

This thesis consists of the introduction (chapter 1), background (chapter 2), four main chapters (chapter 3, 4, 5, 6) each of which covers one research topic, and conclusion (chapter 7). In Chapter 2, we explain some mathematical preliminaries expected to be helpful for those who are not familiar with Riemannian manifold and information geometry. Chapters 3 & 4 present methods to resolve the first issue of the vanilla autoencoder, the "wrong manifold" issue, introduced in Section 1.3.1. Chapter 5 proposes an algorithm to resolve the second issue of the vanilla autoencoder, the "distorted latent space" issue, introduced in Section 1.3.2. In Chapter 6, we apply the algorithm proposed in Chapter 5 to the point cloud data and develop a mathematically necessary tool for this, that is, the Riemannian geometry of the point cloud data space.

Listed below are the publications on which each chapter is based and my personal contributions to each project.

Chapter 1 Introduction. Not published before, and written from scratch for this thesis.

Chapter 2 Mathematical Preliminaries. Not published before, and written from scratch for this thesis.

Chapter 3 is based on:

Yonghyeon Lee, Hyukjun Kwon, and Frank C. Park, *Neighborhood Reconstructing Autoencoders*. Conference on Neural Information Processing Systems (NeurIPS), 2021 [13].

Personal contributions: original idea to define a new neighborhood reconstruction loss by exploiting the higher-order derivatives of the decoder, implementation & a majority of the experiments, and writing.

Chapter 4 Not published before, and written from scratch for this thesis.

Chapter 5 is based on:

Yonghyeon Lee, Sangwoong Yoon, Minjun Son, and Frank C. Park, *Regularized Autoencoders for Isometric Representation Learning*. International Conference on Learning Representation (ICLR), 2022 [14].

Personal contributions: original idea to generalize the distortion minimization framework in [12] to a more relaxed version tailored for autoencoder regularizations, practical implementation & a majority of the experiments, and writing.

Chapter 6 is based on:

Yonghyeon Lee*, Seungyeon Kim*, Jinwon Choi, and Frank C. Park (*: equal contribution), *A Statistical Manifold Framework for Point Cloud Data*. International Conference on Machine Learning (ICML), 2022 [15].

Personal contributions: original idea to use the statistical manifold framework to construct the Riemannian geometric structure for the point cloud data space, and writing. Development of mathematical theory presented in this paper together with Seungyeon Kim. Experiments were mostly performed by Seungyeon Kim.

Chapter 7 Conclusion. Not published before, and written from scratch for this thesis.

2
Mathematical Preliminaries

2.1 Riemannian Manifold

2.1.1 Riemannian Manifolds in $\mathbb{R}^D$

A **manifold** is a mathematical object that formalizes and generalizes the notions of curves and surfaces which correspond to one- and two-dimensional manifolds, respectively. Formally, an m-dimensional manifold is a topological space[1] where each point has a neighborhood that is homeomorphic[2] to an open subset of m-dimensional Euclidean space. One-dimensional manifolds include lines and circles, but not lemniscates (figure-eight or ∞-shaped curves), because, near the intersection point of the lemniscate, it does not look like a one-dimensional Euclidean space (Figure 2.1).

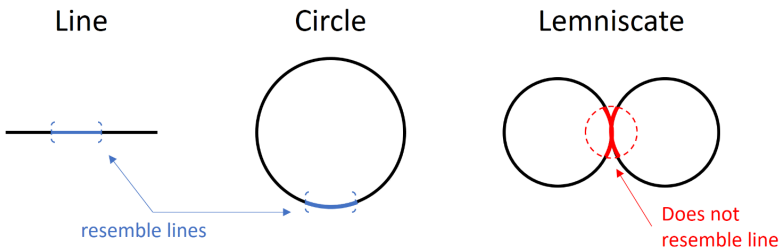


Figure 2.1: Examples of one-dimensional manifold, line and circle, and non-manifold, lemniscate.

Probably, the easiest way to imagine manifolds is to think of them as being lies

[1] A topological space allows for the definition of limits, continuity, and connectedness (e.g., Euclidean space $\mathbb{R}^m$).

[2] Two spaces are homeomorphic if there exists a homeomorphism, a continuous and invertible map, between the two spaces.

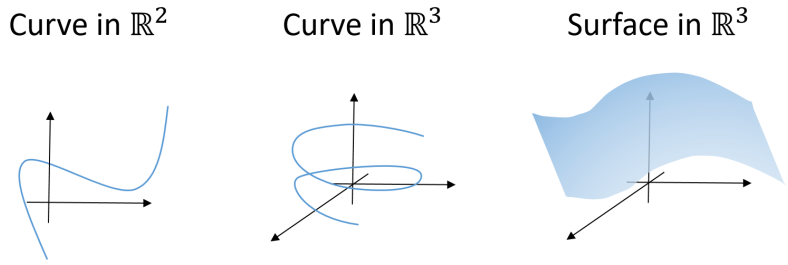


Figure 2.2: Examples of embedded manifolds, curves in two- and three-dimensional Euclidean spaces, and a surface in three-dimensional Euclidean space.

in surrounding Euclidean spaces.[3] Figure 2.2 shows examples of the manifolds embedded in the ambient Euclidean spaces, curves in two- and three-dimensional Euclidean spaces, and a surface in three-dimensional Euclidean space.

We first introduce two ways to (locally) parametrize manifolds $\mathcal{M}$ of dimension m embedded in the ambient Euclidean space $\mathbb{R}^D$ ($D > m$). The manifold $\mathcal{M}$ of dimension m in $\mathbb{R}^D$ can be locally parametrized explicitly in terms of local coordinates $q = (q^1, \cdots, q^m) \in \mathbb{R}^m$ with a coordinate system $\phi : \mathbb{R}^m \to \mathcal{M} \subset \mathbb{R}^D$ such that $x = \phi(q)$. This is called **explicit parametrization**. The other way is to parametrize $\mathcal{M}$ in implicit form by a system of $D - m$ equations of the form $f(x) = 0$ with $f : \mathbb{R}^D \to \mathbb{R}^{D-m}$. This is called **implicit parametrization**.

Remark 1 Differentiable manifold. If a mapping $\phi : \mathbb{R}^m \to \mathbb{R}^D$ is smooth and the $D \times m$ Jacobian matrix $\frac{\partial \phi}{\partial q}(q)$ is of maximal rank m at every point, then the image of ϕ is an m-dimensional differentiable manifold. If a system of $D - m$ equations $f : \mathbb{R}^D \to \mathbb{R}^{D-m}$ is smooth and the $D \times D - m$ Jacobian matrix $\frac{\partial f}{\partial x}$ is of maximal rank $D - m$ at every x that satisfies $f(x) = 0$, then the set $\{x \in \mathbb{R}^D | f(x) = 0\}$ is an m-dimensional differentiable manifold. Manifolds need differentiable structures for useful geometric concepts – which will be introduced later such as the tangent vector, tangent space, Riemannian metric, etc. – to be defined. Examples of non-differentiable manifolds are shown in Figure 2.3.

Example 2.1.1. Sphere manifold S^2. Consider a two-dimensional sphere manifold S^2, a set of all unit vectors in $\mathbb{R}^3$. To explicitly (locally) parametrize the manifold, we can use a spherical coordinate system $(\theta, \phi) \in \mathbb{R}^2 \mapsto (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta) \in$

[3] The Whitney embedding theorem states that any manifold can be embedded in some higher-dimensional Euclidean space.

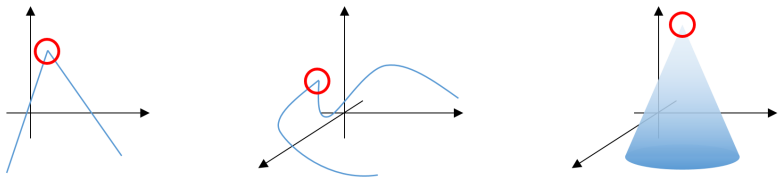


Figure 2.3: Examples of non-differentiable manifolds (not differentiable at the red circled points).

$\mathbb{R}^3$, or to implicitly parametrize the manifold, we can think of the equation $(x^1)^2 + (x^2)^2 + (x^3)^2 - 1 = 0$ for $x = (x^1, x^2, x^3) \in \mathbb{R}^3$.

Example 2.1.2. Manifold of symmetric positive-definite matrices (SPD manifold). Consider a set of $m \times m$ symmetric positive-definite matrices

$$\mathcal{P}(m) := \{P \in \mathbb{R}^{m \times m} | P = P^T, \text{eigenvalues}(P) > 0\}. \tag{2.1.1}$$

This is an $m(m+1)/2$-dimensional manifold embedded in $\mathbb{R}^{m^2}$, where $m^2 - m(m-1)/2$ (symmetry constraints) computes the dimension of the manifold. The $m \times m$ matrix expression can be considered as an implicit parametrization. Consider an $m(m+1)/2$-dimensional vector $v \in \mathbb{R}^{m(m+1)/2}$, by rearranging it to an $m \times m$ symmetric matrix $S(v) \in \mathbb{R}^{m \times m}$ and applying the matrix exponential $\exp(S(v))$, the vector v maps to a symmetric positive-definite matrix; $v \in \mathbb{R}^{m(m+1)/2} \mapsto \exp(S(v)) \in \mathcal{P}(m)$ is an explicit parametrization.

When explicitly parametrizing a manifold that is not homeomorphic to the Euclidean space, e.g., a one-dimensional circular manifold S^1, to globally parametrize the manifold, we need more than one local coordinate system. Considering the circular manifold S^1, we need at least two local coordinate systems to cover the entire manifold as shown in Figure 2.4, and a transition map in the overlapped region can be defined. The family of coordinate systems that covers the manifold is called an **atlas**. Throughout, we will assume the manifold is homeomorphic to the Euclidean space and consider a single coordinate system for simplicity; geometric concepts explained later can be straightforwardly generalized to arbitrary manifolds with multiple coordinate systems.

Let $\mathcal{M}$ be an m-dimensional differentiable manifold in $\mathbb{R}^D$ parametrized in local coordinates q by $x = \phi(q)$. Let $x(t)$ be an arbitrary smooth curve in $\mathcal{M}$ that passes through $\bar{x}$ at $t = 0$, i.e., $x(0) = \bar{x}$, and $q(t) \in \mathbb{R}^m$ be the corresponding curve in local coordinates, i.e., $x(t) = \phi(q(t))$ such that $x(0) = \phi(q(0)) = \phi(\bar{q})$. Differentiating both

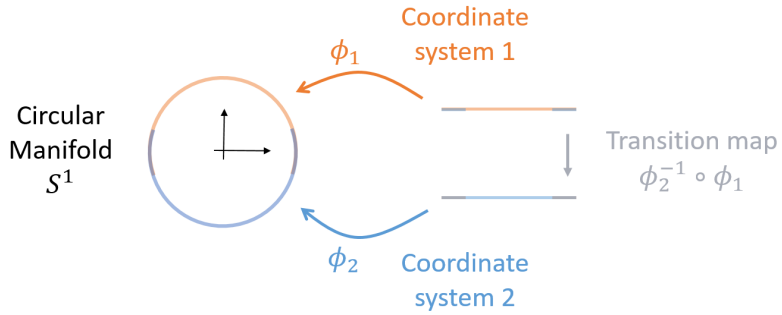


Figure 2.4: Two local coordinate systems $\phi_1, \phi_2 : \mathbb{R}^1 \to S^1 \subset \mathbb{R}^2$ globally parametrize the circular manifold S^1; in the overlapped region, a transition map $\phi_2^{-1} \circ \phi_1 : \mathbb{R}^1 \to \mathbb{R}^1$ is defined.

sides and evaluating at $t = 0$, we have

$$\dot{x}(0) = \sum_{i=1}^{m} \frac{\partial \phi}{\partial q^i}(\bar{q}) \dot{q}^i(0). \tag{2.1.2}$$

We note that the velocity vector $\dot{x}(0)$ cannot be an arbitrary vector in $\mathbb{R}^D$ but must lie in the column space of the matrix $\frac{\partial \phi}{\partial q}(\bar{q})$ (i.e., must be tangent to $\mathcal{M}$ at $\bar{x}$), which is a linear subspace of dimension m. This vector space is called the **tangent space** viewed as being attached at $\bar{x}$ as illustrated in Figure 2.5, and denoted by $T_{\bar{x}}\mathcal{M}$. For any point $x \in \mathcal{M}$, there is a tangent space $T_x\mathcal{M}$ attached at x, and they are all different vector spaces.

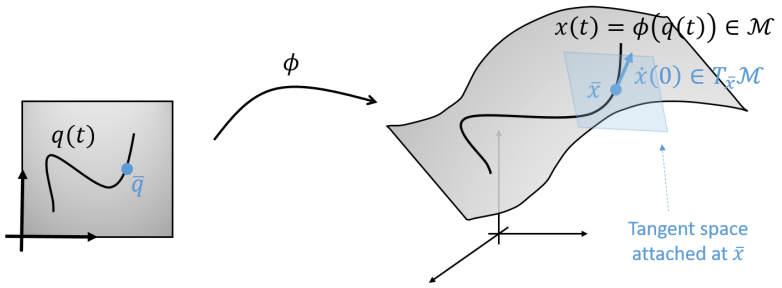


Figure 2.5: Illustration of the tangent space of $\mathcal{M}$ attached at $\bar{x} \in \mathcal{M}$, $T_{\bar{x}}\mathcal{M}$, which is equivalent to the column space of the matrix $\frac{\partial \phi}{\partial q}(\bar{q})$.

In linear algebra, a mathematical object called an inner product can be defined in vector space, which is used to define the length (or norm) of a vector or the angle between the vectors. As a generalization to differentiable manifolds, one can define the length of a curve, the angle between two intersecting curves, and the volume of an area in the manifold by specifying the inner products for all tangent spaces, and the family of the inner products defined for all tangent spaces

$$\{\langle -, - \rangle_x : T_x\mathcal{M} \times T_x\mathcal{M} \to \mathbb{R} | x \in \mathcal{M}\}, \tag{2.1.3}$$

which is assumed to be smoothly varying over $\mathcal{M}$, is called the **Riemmanian metric**. The manifold $\mathcal{M}$ with the Riemannian metric is called the **Riemannian manifold**.

Given an explicit parametrization by $\phi : q \in \mathbb{R}^m \to x \in \mathcal{M} \subset \mathbb{R}^D$, the Riemannian metric can be expressed as $m \times m$ positive-definite matrices in local coordinates q: let $q(t)$ be a curve in the local coordinates and $x(t) = \phi(q(t))$ be the corresponding curve in $\mathcal{M}$, and consider the following inner product

$$\begin{aligned}
\langle \dot{x}, \dot{x} \rangle_x &= \langle \sum_{i=1}^m \frac{\partial \phi}{\partial q^i} \dot{q}^i, \sum_{j=1}^m \frac{\partial \phi}{\partial q^j} \dot{q}^j \rangle_{\phi(q)} \\
&= \sum_{i=1}^m \sum_{j=1}^m \dot{q}^i \dot{q}^j \langle \frac{\partial \phi}{\partial q^i}, \frac{\partial \phi}{\partial q^j} \rangle_{\phi(q)} \\
&= \sum_{i=1}^m \sum_{j=1}^m \dot{q}^i \dot{q}^j g_{ij}(q) \\
&= \dot{q}^T G(q) \dot{q},
\end{aligned} \tag{2.1.4}$$

where $g_{ij}(q) := \langle \frac{\partial \phi}{\partial q^i}, \frac{\partial \phi}{\partial q^j} \rangle_{\phi(q)}$ and $G(q) = (g_{ij}(q))_{i,j=1,\cdots,m}$ that is an $m \times m$ positive-definite matrix. The family of $G(q)$ for $q \in \mathbb{R}^m$ is the Riemannian metric expressed in the local coordinates.

For example, the simplest choice of the Riemannian metric is to inherit the inner product of the ambient Euclidean space $\mathbb{R}^D$ as follows:

$$\langle v, w \rangle_x := v \cdot w \text{ for all } v, w \in T_x\mathcal{M} \subset \mathbb{R}^D, \tag{2.1.5}$$

for all $x \in \mathcal{M}$. Then, we can define the length of a curve $x(t) : t \in [0, 1]$ in $\mathcal{M}$ as follows:

$$\text{Len}(x(t)) := \int_0^1 \sqrt{\langle \dot{x}(t), \dot{x}(t) \rangle_{x(t)}}\, dt = \int_0^1 \sqrt{\dot{x}(t) \cdot \dot{x}(t)}\, dt. \tag{2.1.6}$$

In local coordinates where $x(t) = \phi(q(t))$, the length of $q(t)$ can be expressed as

follows:

$$\text{Len}(q(t)) := \int_0^1 \sqrt{\dot{q}(t)^T G(q(t))\dot{q}(t)}\, dt, \tag{2.1.7}$$

where

$$G(q) := \frac{\partial \phi}{\partial q}(q)^T \frac{\partial \phi}{\partial q}(q) \in \mathbb{R}^{m \times m} \tag{2.1.8}$$

is a symmetric and positive-definite matrix that smoothly varies over $q \in \mathbb{R}^m$. The family of $G(q)$ for $q \in \mathbb{R}^m$ is the Riemannian metric expressed in local coordinates q.

Example 2.1.3. Riemannian metric of sphere manifold S^2. Consider a local coordinate system for the two-dimensional spherical manifold S^2, that is $\psi : (\theta, \phi) \in \mathbb{R}^2 \mapsto (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\phi) \in S^2 \subset \mathbb{R}^3$. The Riemannian metric inherited from the three-dimensional ambient Euclidean space can be expressed in the local coordinates as follows:

$$G(\theta, \phi) = \begin{bmatrix} 1 & 0 \\ 0 & \sin^2\theta \end{bmatrix}. \tag{2.1.9}$$

Then, given a curve in S^2 expressed in local coordinates as $(\theta(t), \phi(t)) : t \in [0, 1]$, the length of it can be computed as follows: $\int_0^1 \sqrt{\dot{\theta}(t)^2 + \dot{\phi}(t)^2 \sin^2\theta(t)} dt$.

Given two points $x_1, x_2 \in \mathcal{M}$, let $q_1, q_2 \in \mathbb{R}^m$ be their local coordinates and $G(q)$ be the Riemannian metric expressed in the local coordinates. The **minimal geodesic** on $\mathcal{M}$ can be defined as the curve of the shortest length connecting the two points with the constant velocity:

$$\min_{q(t)} \int_0^1 \dot{q}(t)^T G(q(t))\dot{q}(t)\, dt \quad \text{s.t.} \quad q(0) = q_1, q(1) = q_2. \tag{2.1.10}$$

The solution curve is also a minimizer of $\text{Len}(q(t))$. Then the **geodesic distance** between $x_1, x_2 \in \mathcal{M}$ is defined as the length of the minimal geodesic curve.

Consider an open subset $U \subset \mathcal{M}$ and the corresponding subset $V \subset \mathbb{R}^m$ in local coordinates such that $U = \phi(V)$. The volume of U can be computed in the coordinates as follows:

$$\text{Vol}(V) := \int_V \sqrt{\det(G(q))} dq^1 \cdots dq^m, \tag{2.1.11}$$

where $\sqrt{\det(G(q))}$ compensates for any distortion in the infinitesimal volume $dq^1 \cdots dq^m$ when mapped to the manifold $\mathcal{M}$, $\sqrt{\det(G(q))} dq^1 \cdots dq^m$ is called the **Riemannian volume form**.

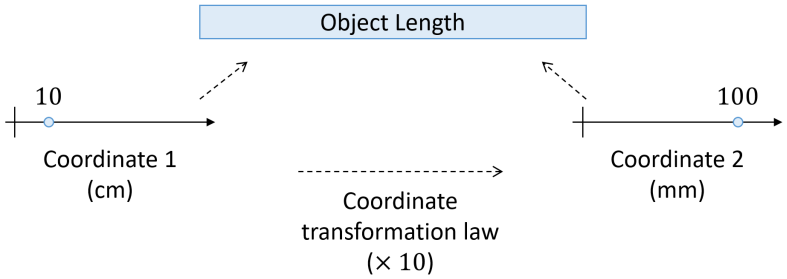


Figure 2.6: The length of an object can have different numerical representations depending on the choice of local coordinates (cm or mm).

2.1.2 Intrinsic Definition of Riemannian Manifolds *

In this section, we revisit the notion of the Riemannian manifold from the intrinsic point of view without involving any embedding in $\mathbb{R}^D$. To parametrize the manifold, one needs to set a local coordinate system (just as the explicit parametrization introduced in the previous section), however, all the constructs – minimal geodesics, volume forms, etc. – should not depend on the choice of local coordinates, but only on the structure of the manifold and the choice of Riemannian metric. All geometric concepts on the manifold should be defined intrinsically without involving any coordinate system, and once local coordinates are introduced, they have the corresponding numerical representations in these local coordinates. Given another choice of local coordinates, the numerical representations become different, and there exist proper coordinate transformation laws. For example, when representing the length of an object, the number depends on which unit you use, cm or mm, and there is a transformation law (Figure 2.6). Throughout this section, we will focus on providing coordinate-free formulations of the geometric notions such as the tangent space and Riemannian metric and their coordinate transformation laws.

For a differentiable manifold $\mathcal{M}$, let $\psi : \mathcal{M} \to \mathbb{R}^m$ be a homeomorphism (we assume $\mathcal{M}$ is homeomorphic to $\mathbb{R}^m$ for simplicity). To define the tangent vector and tangent space in a coordinate-free manner, we use a real-valued function $f : \mathcal{M} \to \mathbb{R}$ whose coordinate representation $f \circ \psi^{-1} : \mathbb{R}^m \to \mathbb{R}$ is infinitely differentiable (the set of all such functions is denoted by $C^\infty(\mathcal{M})$). Intuitively, given a direction vector v (here v is an abstract symbol) tangent to the manifold at $\mathrm{x} \in \mathcal{M}$, as a generalization of the directional derivative from vector calculus, we can think of the rate at which the function f changes at a point x in the direction v. A **derivation** at $\mathrm{x} \in \mathcal{M}$ (in v)

is defined as a linear map $D_{\mathrm{v}}(\cdot)|_{\mathrm{x}} : C^\infty(\mathcal{M}) \to \mathbb{R}$ that satisfies the Leibniz identity:

$$\forall f, g \in C^\infty(\mathcal{M}) \quad D_{\mathrm{v}}(fg)|_{\mathrm{x}} = D_{\mathrm{v}}(f)|_{\mathrm{x}} \cdot g(\mathrm{x}) + f(\mathrm{x}) \cdot D_{\mathrm{v}}(g)|_{\mathrm{x}}, \tag{2.1.12}$$

which is modeled on the product rule of calculus. Setting

$$(D_{\mathrm{v}} + D_{\mathrm{w}})(f)|_{\mathrm{x}} := D_{\mathrm{v}}(f)|_{\mathrm{x}} + D_{\mathrm{w}}(f)|_{\mathrm{x}} \;\; \& \;\; (\lambda \cdot D_{\mathrm{v}})(f)|_{\mathrm{x}} := \lambda \cdot D_{\mathrm{v}}(f)|_{\mathrm{x}}, \tag{2.1.13}$$

the set of all derivations at x turns into a vector space, which is defined to be the **tangent space** attached at $\mathrm{x} \in \mathcal{M}$ denoted by $T_{\mathrm{x}}\mathcal{M}$. A derivation $D_{\mathrm{v}}(\cdot)|_{\mathrm{x}}$ becomes a **tangent vector** at $\mathrm{x} \in \mathcal{M}$. We note that no local coordinates are introduced in these definitions.

Let x be the coordinates of $\mathrm{x} \in \mathcal{M}$, i.e., $\psi(\mathrm{x}) = x$, then a tangent vector or derivation at x can be written in the local coordinates as follows:

$$D_{\mathrm{v}}(\cdot)|_{\mathrm{x}} = \sum_{i=1}^{m} v^i \frac{\partial}{\partial x^i}((\cdot) \circ \psi^{-1})|_x, \tag{2.1.14}$$

where $v = (v^1, \cdots, v^m) \in \mathbb{R}^m$ is the coordinate representation of the tangent vector, and $\frac{\partial}{\partial x^i}((\cdot) \circ \psi^{-1})|_x$ or shortly $\frac{\partial}{\partial x^i}$ is often used to denote the i-th basis vector for $T_{\mathrm{x}}\mathcal{M}$ (the tangent vector is often simply denoted by $\sum_{i=1}^m v^i \frac{\partial}{\partial x^i}$). Now, consider a different coordinate system $\tilde{\psi} : \mathcal{M} \to \mathbb{R}^m$ and denote the corresponding coordinates by $\tilde{x} = \tilde{\psi}(\mathrm{x})$; then $\psi \circ \tilde{\psi}^{-1} : \tilde{x} \in \mathbb{R}^m \mapsto x \in \mathbb{R}^m$ is the coordinate transformation. Let $\tilde{v} \in \mathbb{R}^m$ be the coordinate representation of the tangent vector $D_{\mathrm{v}}(\cdot)|_{\mathrm{x}}$ in coordinates $\tilde{x}$, then the following equalities hold:

$$\begin{aligned} D_{\mathrm{v}}(f)|_{\mathrm{x}} = \sum_{i=1}^{m} v^i \frac{\partial}{\partial x^i}(f \circ \psi^{-1})|_x &= \sum_{j=1}^{m} \tilde{v}^j \frac{\partial}{\partial \tilde{x}^j}(f \circ \tilde{\psi}^{-1})|_{\tilde{x}} \\ &= \sum_{j=1}^{m} \tilde{v}^j \frac{\partial}{\partial \tilde{x}^j}(f \circ \psi^{-1} \circ \psi \circ \tilde{\psi}^{-1})|_{\tilde{x}} \\ &= \sum_{i=1}^{m} \sum_{j=1}^{m} \tilde{v}^j \frac{\partial}{\partial \tilde{x}^j}(\psi \circ \tilde{\psi}^{-1})^i|_{\tilde{x}} \frac{\partial}{\partial x^i}(f \circ \psi^{-1})|_x. \end{aligned} \tag{2.1.15}$$

Therefore, the coordinate transformation law between v and $\tilde{v}$ is given as follows:

$$v^i = \sum_{j=1}^{m} \tilde{v}^j \frac{\partial}{\partial \tilde{x}^j} (\psi \circ \tilde{\psi}^{-1})^i|_{\tilde{x}} = \sum_{j=1}^{m} \tilde{v}^j \frac{\partial x^i}{\partial \tilde{x}^j}, \tag{2.1.16}$$

where $\frac{\partial x^i}{\partial \tilde{x}^j}$ denotes $\frac{\partial}{\partial \tilde{x}^j}(\psi \circ \tilde{\psi}^{-1})^i|_{\tilde{x}}$.

Consider a symmetric bilinear form (or inner product) on the tangent vector space $T_{\mathrm{x}}\mathcal{M}$ and denote it by $\langle \cdot, \cdot \rangle_{\mathrm{x}} : T_{\mathrm{x}}\mathcal{M} \times T_{\mathrm{x}}\mathcal{M} \to \mathbb{R}$. The **Riemannian metric** is a family of the inner products $\{\langle \cdot, \cdot \rangle_{\mathrm{x}} | \mathrm{x} \in \mathcal{M}\}$ that smoothly varies over $\mathcal{M}$. We note that this is again defined in a coordinate-free way. Given local coordinates $x = \psi(\mathrm{x})$, consider two tangent vectors $\sum_{i=1}^{m} v^i \frac{\partial}{\partial x^i}, \sum_{j=1}^{m} w^j \frac{\partial}{\partial x^j} \in T_{\mathrm{x}}\mathcal{M}$. The inner product between these two vectors can be written as follows:

$$\langle \sum_{i=1}^{m} v^i \frac{\partial}{\partial x^i}, \sum_{j=1}^{m} w^j \frac{\partial}{\partial x^j} \rangle_{\mathrm{x}} = \sum_{i=1}^{m} \sum_{j=1}^{m} v^i w^j \langle \frac{\partial}{\partial x^i}, \frac{\partial}{\partial x^j} \rangle_{\mathrm{x}}. \tag{2.1.17}$$

If we denote $\langle \frac{\partial}{\partial x^i}, \frac{\partial}{\partial x^j} \rangle_{\mathrm{x}}$ by $g_{ij}(x)$, then the inner product is expressed as follows:

$$\sum_{i=1}^{m} \sum_{j=1}^{m} v^i w^j g_{ij}(x) = v^T G(x) w, \tag{2.1.18}$$

where $G(x) \in \mathbb{R}^{m \times m}$ is the symmetric positive-definite matrix whose (i, j)-th element is $g_{ij}(x)$. With the Riemannian metric expressed in the local coordinates $G(x)$, all the constructs introduced in the previous chapter – minimal geodesics, volume forms, etc. – can be computed using the same formulas.

Lastly, we introduce a coordinate transformation law for the Riemanian metric $g_{ij}(x)$ or $G(x)$. Let $\tilde{x} = \tilde{\psi}(\mathrm{x})$ be another coordinate system and $\tilde{g}_{ab}(\tilde{x})$ and $\tilde{G}(\tilde{x})$ be the corresponding coordinate representations of the Riemannian metric, given two tangent vectors expressed in each coordinate as v, w and $\tilde{v}, \tilde{w}$, the following equalities

hold:

$$\begin{aligned}\sum_{i=1}^{m}\sum_{j=1}^{m} v^i w^j g_{ij}(x) &= \sum_{a=1}^{m}\sum_{b=1}^{m} \tilde{v}^a \tilde{w}^b \tilde{g}_{ab}(\tilde{x}) \\ &= \sum_{a=1}^{m}\sum_{b=1}^{m}\sum_{i=1}^{m}\sum_{j=1}^{m} v^i \frac{\partial \tilde{x}^a}{\partial x^i} w^j \frac{\partial \tilde{x}^b}{\partial x^j} \tilde{g}_{ab}(\tilde{x}) \\ &= \sum_{i=1}^{m}\sum_{j=1}^{m} v^i w^j (\sum_{a=1}^{m}\sum_{b=1}^{m} \frac{\partial \tilde{x}^a}{\partial x^i} \frac{\partial \tilde{x}^b}{\partial x^j} \tilde{g}_{ab}(\tilde{x})), \end{aligned} \tag{2.1.19}$$

and thus

$$g_{ij}(x) = \sum_{a=1}^{m}\sum_{b=1}^{m} \frac{\partial \tilde{x}^a}{\partial x^i} \frac{\partial \tilde{x}^b}{\partial x^j} \tilde{g}_{ab}(\tilde{x}). \tag{2.1.20}$$

In the matrix representation,

$$G(x) = (\frac{\partial \tilde{x}}{\partial x})^T \tilde{G}(\tilde{x}) (\frac{\partial \tilde{x}}{\partial x}), \tag{2.1.21}$$

where $\frac{\partial \tilde{x}}{\partial x} \in \mathbb{R}^{m\times m}$ is the invertible matrix whose (a,i)-th element is $\frac{\partial \tilde{x}^a}{\partial x^i}$.

Example 2.1.4. Grassmannian manifold The set of all k-dimensional linear subspaces in $\mathbb{R}^n$ is called the Grassmannian manifold of k-planes in $\mathbb{R}^n$ and denoted by $\mathrm{GR}_k(\mathbb{R}^n)$. One may want to represent k-planes in $\mathbb{R}^n$ as the column spaces of $n \times k$ matrices $X \in \mathbb{R}^{n\times k}$ of rank k and think of the Grassmannian manifold as an nk-dimensional manifold, but this representation is not one-to-one, i.e., there are many $n \times k$ matrices (indeed infinitely many) that have the same column space – for any $k \times k$ invertible matrix $I \in \mathbb{R}^{k\times k}$, the matrices X and $XI \in \mathbb{R}^{n\times k}$ have the same column space. Therefore, when considering the dimension of $\mathrm{GR}_k(\mathbb{R}^n)$, the dimension of the set of all $k \times k$ invertible matrices must be subtracted from the dimension of the set of all $n \times k$ matrices of rank k, and thus it is an $nk - k^2$-dimensional (more precisely, $\mathrm{GR}_k(\mathbb{R}^n)$ is a quotient of the set of all $n \times k$ matrices of rank k by the set of all invertible $k \times k$ matrices).

Now we describe one choice of local coordinate system. For $n \times k$ matrix whose range in an element of $\mathrm{GR}_k(\mathbb{R}^n)$, i.e., has a maximal rank k, we can apply elementary

column operations to obtain its reduced column echelon form:

$$\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ a_{1,1} & \cdots & \cdots & a_{1,k} \\ \vdots & & & \vdots \\ a_{n-k,1} & \cdots & \cdots & a_{n-k,k} \end{bmatrix}, \tag{2.1.22}$$

where the $n-k \times k$ matrix $A = (a_{i,j})_{i=1,\cdots,n-k \,\&\, j=1,\cdots,k}$ determines which element it is in the manifold.

2.1.3 Mappings between Riemannian Manifolds

In this section, we introduce an isometry – which preserves the geometric quantities such as the distances between points, angles between curves, and volumes of regions – between Riemannian manifolds, and the Riemannian distortion measure that measures how far a mapping from being an isometry, which is particularly relevant to Chapter 5. This section summarizes Chapter 2 of Jang's thesis "Riemannian Distortion Measures for Non-Euclidean Data" [12], please refer to the original paper for details.

Let $\mathcal{M}$ be an m-dimensional Riemannian manifold with local coordinates $x = (x^1, \cdots, x^m)$ and Riemannian metric $g_{ij}(x)$ or $G(x)$, and $\mathcal{N}$ be an n-dimensional Riemannian manifold with local coordinates $y = (y^1, \cdots, y^n)$ and Riemannian metric $h_{ab}(y)$ or $H(y)$. Consider a mapping $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ between two Riemannian manifolds $\mathcal{M}$ and $\mathcal{N}$, which maps a point x, a curve $\mathcal{C}$, and a region $\mathcal{V}$ to $\mathrm{f(x)}$, $\mathrm{f}(\mathcal{C})$, and $\mathrm{f}(\mathcal{V})$ as shown in Figure 2.7. Let $x(t) : t \in [0,1]$ represent the curve $\mathcal{C}$ and V represent the region $\mathcal{V}$ in the local coordinates of $\mathcal{M}$ and $f : \mathbb{R}^m \to \mathbb{R}^n$ be the coordinate representation of the mapping f, then the mapped curve and region are represented as $y(t) = f(x(t)) : t \in [0,1]$ and $f(V)$ in the local coordinates of $\mathcal{N}$.

To introduce the conditions for the mapping f to being an isometry, let's compare the length of a curve $\mathcal{C}$ in $\mathcal{M}$ and that of $\mathrm{f}(\mathcal{C}) \in \mathcal{N}$. The lengths of both curves are

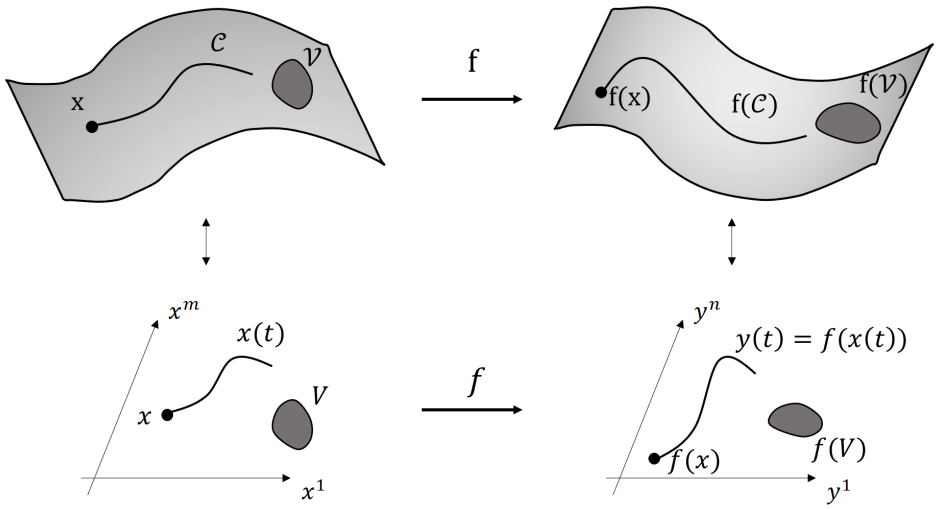


Figure 2.7: A mapping $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ and its coordinate representation $f : \mathbb{R}^m \to \mathbb{R}^n$.

computed as follows:

$$\mathrm{Len}(\mathcal{C}) = \int_0^1 \sqrt{\dot{x}^T G(x)\dot{x}}\, dt \tag{2.1.23}$$

$$\begin{aligned}\mathrm{Len}(\mathrm{f}(\mathcal{C})) &= \int_0^1 \sqrt{\dot{y}^T H(y)\dot{y}}\, dt \\ &= \int_0^1 \sqrt{\dot{x}^T \frac{\partial f}{\partial x}(x)^T H(f(x)) \frac{\partial f}{\partial x}(x)\dot{x}}\, dt \\ &= \int_0^1 \sqrt{\dot{x}^T J_f(x)^T H(f(x)) J_f(x)\dot{x}}\, dt, \end{aligned} \tag{2.1.24}$$

where $J_f(x) = \frac{\partial f}{\partial x}(x) \in \mathbb{R}^{n\times m}$ (the differential of f at x expressed in local coordinates) and we omit writing (t) in the integrands for better readability (e.g., $x(t) \mapsto x$).

For the lengths of the two curves to be identical for any curve $\mathcal{C}$ in $\mathcal{M}$, the following equality should hold:

$$G(x) = J_f(x)^T H(f(x)) J_f(x) \quad \text{for all} \quad \mathrm{x} \in \mathcal{M}, \tag{2.1.25}$$

where x denotes the coordinate representation of x, and in this case, f is called an **isometry**. Note that, if f is an isometry, the volume of a region $\mathcal{V}$ in $\mathcal{M}$ and that of

$\mathrm{f}(\mathcal{V})$ in $\mathcal{N}$ are also identical:

$$\mathrm{Vol}(\mathcal{V}) = \int_V \sqrt{\det(G(x))}\, dx \tag{2.1.26}$$

$$\begin{aligned}\mathrm{Vol}(\mathrm{f}(\mathcal{V})) &= \int_{f(V)} \sqrt{\det(H(y))}\, dy \\ &= \int_V \sqrt{\det(H(f(x)))}\, |\det(J_f(x))|\, dx \\ &= \int_V \sqrt{\det(H(f(x)))\det(J_f(x))^2}\, dx \\ &= \int_V \sqrt{\det(J_f(x)^T H(f(x)) J_f(x))}\, dx. \end{aligned} \tag{2.1.27}$$

The matrix $J_f^T(x)H(f(x))J_f(x) \in \mathbb{R}^{m\times m}$ is referred to as the pullback metric of H via the mapping f. The pullback metric may become positive-semidefinite, e.g., when $m > n$, an isometry is unachievable in such cases.

Now, we introduce how to formulate **global geometric distortion measures** for a smooth mapping $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ which measures how far the mapping f from being an isometry. The measure should be defined intrinsically, i.e., in a coordinate-invariant way. Denote by $\mathrm{y} = \mathrm{f}(\mathrm{x})$ and $y = f(x)$ in local coordinates. At a point $\mathrm{x} \in \mathcal{M}$ denoted by x in local coordinates, let $\lambda_i, i = 1, \cdots, m$ be the eigenvalues of $J_f(x)^T H((f(x)))J_f(x)G(x)^{-1}$. Apart from their order, a set of them $\{\lambda_i\}_{i=1}^m$ is intrinsically associated with $J_f^T H J_f$ and G, meaning that it is invariant under a pair of coordinate transformations $x \mapsto \tilde{x} = \phi(x)$ and $y \mapsto \tilde{y} = \psi(y)$. The metrics G, H, and the Jacobian J_f transform according to the following rules: (i) $G \mapsto \tilde{G} = \Phi^{-T} G \Phi^{-1}$, where $\Phi = \frac{\partial \phi}{\partial x}$; (ii) $H \mapsto \tilde{H} = \Psi^{-T} H \Psi^{-1}$, where $\Psi = \frac{\partial \psi}{\partial y}$; (iii) $J_f \mapsto \tilde{J}_{\tilde{f}} = \Psi J_f \Phi^{-1}$; and thus

$$\begin{aligned} J_f^T H J_f G^{-1} \mapsto \tilde{J}_{\tilde{f}}^T \tilde{H} \tilde{J}_{\tilde{f}} \tilde{G}^{-1} &= (\Psi J_f \Phi^{-1})^T (\Psi^{-T} H \Psi^{-1}) \Psi J_f \Phi^{-1} (\Phi^{-T} G \Phi^{-1})^{-1} \\ &= \Phi^{-T} J_f^T H J_f G^{-1} \Phi^T; \end{aligned} \tag{2.1.28}$$

it is then straightforward to verify that the eigenvalues remain the same (To see why, observe that $\det(\Phi A \Phi^{-1} - \lambda I) = \det(\Phi A \Phi^{-1} - \lambda \Phi \Phi^{-1}) = \det(\Phi(A - \lambda I)\Phi^{-1}) = \det(A - \lambda I)$).

Let $\sigma(\lambda_1, \cdots, \lambda_m)$ be any symmetric function (i.e., a function whose value is invariant with respect to permutations of its arguments) of the m eigenvalues, then the

integral

$$\int_{\mathcal{M}} \sigma(\lambda_1, \cdots, \lambda_m)\sqrt{\det G}\, dx^1 \cdots dx^m \tag{2.1.29}$$

is an intrinsic quantity, i.e., coordinate-invariant. We now examine choices for σ that capture the intrinsic distortion of the mapping f. The ideal case of no distortion is achieved when f is an isometry, i.e., $J_f^T H J_f G^{-1} = I$ or $\lambda_i = 1, i = 1, \cdots, m$ at every $\mathrm{x} \in \mathcal{M}$. Therefore, one straightforward choice for $\sigma(\lambda)$ that captures the intrinsic distortion is

$$\sigma(\lambda) = \frac{1}{2}\sum_{i=1}^{m}(1-\lambda_i)^2. \tag{2.1.30}$$

There are other choices as well, we refer to [12].

2.2 Information Geometry

2.2.1 Statistical Manifold

A **statistical manifold** is a manifold, each of whose points is a probability distribution. Consider a set of parametric probability density functions

$$\mathcal{S} := \{p(x;\theta)|\theta \in \Theta\}, \tag{2.2.31}$$

where $\Theta \subset \mathbb{R}^m$ and $p(x;\theta)$ denotes a probability density function in $\mathbb{R}^n$ with the parameters $\theta = (\theta^1, \cdots, \theta^m) \in \Theta$. If a mapping that maps the parameter vector to the corresponding density function $p(x;\cdot) : \theta \mapsto p(x;\theta)$ is injective (i.e., if $p(x;\theta_1) = p(x;\theta_2)$, then $\theta_1 = \theta_2$) and smooth, then the set of density functions $\mathcal{S}$ can be viewed as an m-dimensional differentiable manifold with local coordinates θ.

Example 2.2.1. Manifold of 1-d Gaussian distributions. Consider a set of 1-dimensional Gaussian distributions $\mathcal{S} := \{p(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)|\mu \in \mathbb{R}, \sigma \in \mathbb{R}_{>0}\}$ with parameters μ, σ. It is straightforward that the parametrization is injective and smooth, and thus the set $\mathcal{S}$ is a two-dimensional differentiable manifold with local coordinates $(\mu,\sigma) \in \mathbb{R}^2$ (Figure 2.8).

Example 2.2.2. Manifold of multivariate Gaussian distributions. Consider a set of n-dimensional multivariate Gaussian distributions

$$\mathcal{N}(n) := \{p(x;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}}\exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))|\mu \in \mathbb{R}^n, \Sigma \in \mathcal{P}(n)\}, \tag{2.2.32}$$

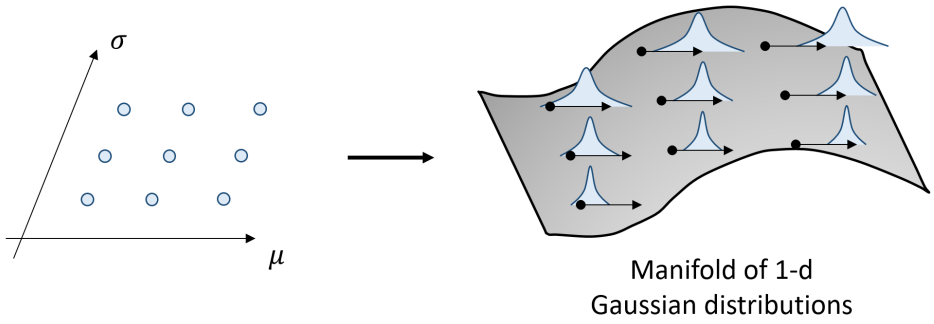


Figure 2.8: Illustration of the two-dimensional manifold of 1-dimensional Gaussian distributions.

where $\mathcal{P}(n)$ is the manifold of symmetric positive-definite matrices from example 2.1.2; the set $\mathcal{N}(n)$ is a differentiable manifold of dimension $n+n(n+1)/2$. A natural choice of local coordinates is given by $(\mu^i)_{i=1,\cdots,n}$ and $(\sigma^{ij})_{i\geq j}$ where $\mu = (\mu^1, \cdots, \mu^m)$ and $\Sigma = (\sigma^{ij})_{i,j=1,\cdots,n}$.

2.2.2 Fisher Information Metric

Let $\mathcal{S} = \{p(x;\theta)|\theta \in \Theta \subset \mathbb{R}^m\}$ be the statistical manifold with coordinates $\theta = (\theta^1, \cdots, \theta^m)$ for $x \in \mathbb{R}^n$, then the **Fisher information metric** that takes the form

$$g_{ij}(\theta) := \int_{\mathbb{R}^n} \frac{\partial \log p(x;\theta)}{\partial \theta^i} \frac{\partial \log p(x;\theta)}{\partial \theta^j} \, p(x;\theta) \, dx^1 \cdots dx^n \tag{2.2.33}$$

acts as a natural Riemannian metric for the statistical manifold. An equivalent form of the above definition is:

$$g_{ij}(\theta) = \int_{\mathbb{R}^n} -\frac{\partial^2}{\partial \theta^i \partial \theta^j} \log p(x;\theta) \, p(x;\theta) \, dx. \tag{2.2.34}$$

To show that this equivalence, note that $\mathbb{E}_{x\sim p(x;\theta)}[\frac{\partial \log p(x;\theta)}{\partial \theta^i}] = 0$ and apply $\frac{\partial}{\partial \theta^j}$ on both sides. We note that the Fisher information metric in invariant under the coordinate transformation $x \mapsto x' = \phi(x)$ which transforms $p(x;\theta) \mapsto p'(x';\theta) = p(\phi^{-1}(x');\theta)|\frac{\partial \phi^{-1}}{\partial x'}(x')|$, because $p(x;\theta)dx = p'(x';\theta)dx'$ and $\log p'(x';\theta) = \log p(x;\theta) + \log|\frac{\partial \phi^{-1}}{\partial x'}(x')|$.

Consider a curve $\theta(t) : t \in [0,1]$ and the corresponding curve of density functions $p(x;\theta(t))$ in the statistical manifold, the length of this curve is then computed as

follows:

$$\begin{aligned}
\mathsf{Len}(\theta(t)) &= \int_0^1 \sqrt{\sum_{i=1}^m \sum_{j=1}^m g_{ij}(\theta(t))\, \dot{\theta}^i(t)\dot{\theta}^j(t)}\, dt \\
&= \int_0^1 \sqrt{\int_{\mathbb{R}^n} \big(\sum_{i=1}^m \dot{\theta}^i(t) \frac{\partial \log p(x;\theta(t))}{\partial \theta^i}\big)^2 p(x;\theta(t))\, dx dt} \\
&= \int_0^1 \sqrt{\int_{\mathbb{R}^n} \big(\frac{d}{dt} \log p(x;\theta(t))\big)^2 p(x;\theta(t))\, dx dt} \\
&= \int_0^1 \sqrt{\mathbb{E}_{x\sim p(x;\theta)}[\big(\frac{d}{dt} \log p(x;\theta(t))\big)^2]}\, dt. \qquad (2.2.35)
\end{aligned}$$

The length is computed as an integral of the expected magnitude of the time derivative of log probability density that measures how much the density function has changed over time.

One of the most well-known and interesting properties of the Fisher information metric is that it approximates the KL-divergence up to the second order as follows:

$$\begin{aligned}
D_{\mathsf{KL}}(p(x;\theta)|p(x;\theta+d\theta)) &= \mathbb{E}_{x\sim p(x;\theta)}[\log p(x;\theta) - \log p(x;\theta+d\theta)] \\
&= \mathbb{E}_{x\sim p(x;\theta)}[-\sum_{i=1}^m d\theta^i \frac{\partial}{\partial \theta^i} \log p(x;\theta) \\
&\qquad - \frac{1}{2}\sum_{i,j=1}^m \frac{\partial^2}{\partial\theta^i \partial\theta^j} \log p(x;\theta) d\theta^i d\theta^j] \\
&= \mathbb{E}_{x\sim p(x;\theta)}[-\frac{1}{2}\sum_{i,j=1}^m \frac{\partial^2}{\partial\theta^i \partial\theta^j} \log p(x;\theta) d\theta^i d\theta^j] \\
&= \frac{1}{2} g_{ij}(\theta) d\theta^i d\theta^j, \qquad (2.2.36)
\end{aligned}$$

where we use $\mathbb{E}_{x\sim p(x;\theta)}[\frac{\partial}{\partial\theta^i} \log p(x;\theta)] = 0$.

Example 2.2.3. Riemannian manifold of multivariate Gaussian distributions. The manifold of multivariate Gaussian distributions $\mathcal{N}(n)$ from example 2.2.2 can be turned into a Riemannian manifold with the Fisher information metric. In terms of the implicit parametrization $(\mu, \Sigma) \in \mathbb{R}^m \times \mathcal{P}(n)$, given two tangent vectors $V = (V_\mu, V_\Sigma)$ and $W = (W_\mu, W_\Sigma)$ (where V_Σ and W_Σ are $n \times n$ symmetric matrices) defined at a

point (μ, Σ), the inner product is given by

$$\langle V, W \rangle_{(\mu,\Sigma)} = V_\mu^T \Sigma^{-1} W_\mu + \frac{1}{2} \mathrm{Tr}(\Sigma^{-1} V_\Sigma \Sigma^{-1} W_\Sigma). \tag{2.2.37}$$

The geodesic curve and distance between two multivariate Gaussian distributions can be computed by solving a pair of partial differential equations called the geodesic equation, we refer to [16] for details.

3

Neighborhood Reconstructing Autoencoders

3.1 Introduction

Autoencoders are widely used to identify, and to generate samples from, the underlying low-dimensional manifold structure of a given data distribution [9, 17]. It has been widely observed that vanilla autoencoders quite often produce manifolds that (i) are highly sensitive to noisy training data (see Figure 3.1(a)), or (ii) have the wrong local connectivity and geometry (see Figure 3.1(b)), significantly impairing their performance. Regularization techniques have had some success in mitigating the former, e.g., the Denoising Autoencoder [18], which uses deliberately corrupted inputs to train the autoencoder, typically learns manifolds that are robust to noise, but not always with the correct local geometry.

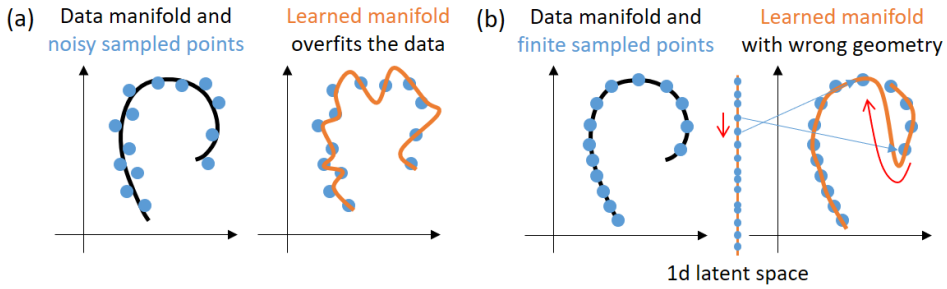


Figure 3.1: Learned manifolds that (a) overfit the data or (b) have the wrong local geometry.

Recently, autoencoder regularization methods that use neighborhood graphs have had some success in addressing the incorrect connectivity issue [19, 20, 21, 22]. Notwithstanding the additional computational overhead of constructing local neighborhood

graphs, the local geometric information obtained from these graphs can significantly reduce errors in the local geometry of the learned manifold, and make learning more well-behaved and robust.

The underlying premise behind these methods is that since the local geometry and topology of the data is captured in the latent space distribution, which is determined entirely by the encoder, regularizing only the encoder should be sufficient; little if any consideration needs to be given to the decoder. The flaw with this premise is that although the encoder learns the correct latent space representation of the manifold data, the decoder is still susceptible to overfitting of the type shown in Figure 3.1(a). These existing methods moreover rely on computation-intensive preprocessing steps like manifold learning [21], linear coefficients computation [22], or computing topological features using persistent homology at each training iteration [19, 23], each of whose computational requirements can grow significantly with problem dimension and scale.

The main contribution is a new graph-based autoencoder training method that addresses both the overfitting and connectivity issues illustrated in Figure 3.1(a)-(b). Like current methods, our method also employs local graphs that capture the local geometry of the data distribution. The key idea behind our method, which we call the **Neighborhood Reconstructing Autoencoder (NRAE)**, is to employ a local quadratic (and in some cases linear) approximation of the decoder function to formulate a new **neighborhood reconstruction loss** in lieu of the point reconstruction loss typically used for autoencoder training. This idea leads to learning the correct geometry and reducing noise sensitivity, significantly improving the robustness of autoencoder training.

To make things more explicit, let $g_\phi : \mathbb{R}^n \to \mathbb{R}^m$ be the encoder (parametrized by ϕ) and $f_\theta : \mathbb{R}^m \to \mathbb{R}^n$ be the decoder (parametrized by θ). Whereas vanilla autoencoders are trained to minimize the sum of the point reconstruction errors $\sum_i \|x_i - f_\theta(g_\phi(x_i))\|^2$, NRAE minimizes a reconstruction error of the form

$$\sum_i \sum_{x \in \mathcal{N}(x_i)} \left\| x - \tilde{f}_\theta(g_\phi(x); g_\phi(x_i)) \right\|^2, \tag{3.1.1}$$

where $\mathcal{N}(x_i)$ is the set of neighborhood points of x_i (including x_i) and $\tilde{f}_\theta(\cdot; g_\phi(x_i))$ is a local quadratic (or linear) approximation of f_θ about $g_\phi(x_i)$. The vanilla autoencoder is obtained by setting $\mathcal{N}(x_i) = \{x_i\}$ for all i. The key idea here is to locally approximate the decoder only, and to exploit the local geometric information extracted from the decoded manifold represented by the image of f_θ.

Like other neighborhood graph-based methods, NRAE also learns the correct local geometry of the decoded manifold. At the same time, the local quadratic (or linear) approximation of f_θ considerably reduces any overfitting to noisy training data or sensitivity to outliers, while maintaining computational efficiency – rather than the entire Jacobian or Hessian of f_θ, only the more easily computed Jacobian-vector and Hessian-vector products are needed for the approximation.

Compared to existing graph-based autoencoder regularization methods, NRAE is easy to implement, computationally efficient, and scalable, requiring only a single prior construction of the graph without additional pre-processing steps. Experiments with both synthetic and image data (*MNIST, Fashion-MNIST, KMNIST, Omniglot, SVHN, CIFAR10, CIFAR100, CelebA*) confirm that overall our method better learns the correct geometry of manifolds, showing improved generalization performance vis-á-vis existing graph-based and other autoencoder regularization methods.

3.2 Neighborhood Reconstructing Autoencoder

In this section, we first provide a high-level mathematical description of the Neighborhood Reconstructing Autoencoder (NRAE), followed by algorithmic details and a discussion of the NRAE's properties and behavior. Throughout we consider a deterministic autoencoder with an encoder function $g_\phi : \mathbb{R}^n \to \mathbb{R}^m$ and decoder function $f_\theta : \mathbb{R}^m \to \mathbb{R}^n$ $(m \leq n)$, with their composition denoted by $F_{\theta,\phi} := f_\theta \circ g_\phi$. We use the notation $\mathcal{D} := \{x_i \in \mathbb{R}^n\}_{i=1}^M$ to denote the set of observed data points.

3.2.1 Mathematical Description

In what follows we use the notation $\mathcal{N}(x)$ to denote the set of neighborhood points of x, with x included in $\mathcal{N}(x)$. We begin with the following definition:

Definition 3.2.1. Let $\tilde{F}_{\theta,\phi}(\cdot; x) := \tilde{f}_\theta(g_\phi(\cdot); g_\phi(x))$, where $\tilde{f}_\theta(\cdot; z)$ is a local quadratic (or in some cases linear) approximation of f_θ at $z = (z^1, z^2, ...z^m)$:

$$\tilde{f}_\theta(z'; z) := f_\theta(z) + \sum_{i=1}^{m} \frac{\partial f_\theta}{\partial z^i}(z) dz^i + \sum_{i,j=1}^{m} \frac{1}{2} \frac{\partial^2 f_\theta}{\partial z^i \partial z^j}(z) dz^i dz^j, \tag{3.2.2}$$

where $dz = z' - z$. $\tilde{F}_{\theta,\phi}(\mathcal{N}(x); x)$ is said to be a **neighborhood reconstruction** of $\mathcal{N}(x)$.

If instead of a quadratic approximation we use the linear approximation of f_θ, the image of $\tilde{F}_{\theta,\phi}(\cdot; x)$ is the tangent space of the decoded manifold at $F_{\theta,\phi}(x)$, and the

neighborhood reconstruction of $\mathcal{N}(x)$ is a subset of the tangent space; the neighborhood reconstruction in this case contains first-order local geometric information about the decoded manifold.

The key idea behind Definition 3.2.1 is that we locally approximate the decoder, and not the encoder, to extract and exploit local geometric information on the decoded manifold, which is captured in the image of $\tilde{F}_{\theta,\phi}(\cdot; x)$ (i.e., the local approximation of the decoded manifold). Figure 3.2 illustrates an example where the autoencoder reconstructs the points almost perfectly, but the neighborhood reconstruction of $\mathcal{N}(x)$, whose elements lie in the tangent space (here we use the linear approximation of f_θ) is considerably different from $\mathcal{N}(x)$.

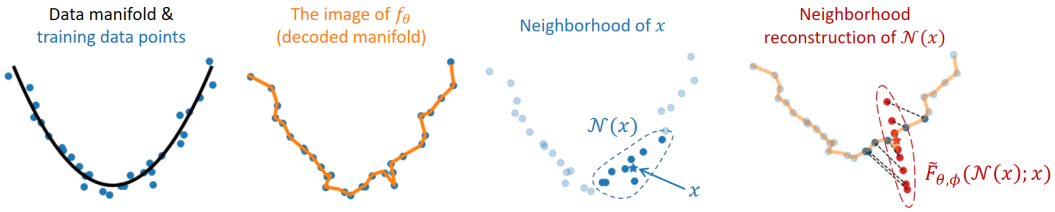


Figure 3.2: The training data points (blue), the decoded manifold (orange), the neighborhood of x denoted by $\mathcal{N}(x)$, and the neighborhood reconstruction (red). The black dotted lines represent the correspondences between $x' \in \mathcal{N}(x)$ and $\tilde{F}_{\theta,\phi}(x'; x)$.

Given that the neighborhood reconstruction of $\mathcal{N}(x)$ reflects the local geometry of the decoded manifold, minimizing a loss function that measures the difference between $\mathcal{N}(x)$ and its image $\tilde{F}_{\theta,\phi}(\mathcal{N}(x); x)$ is one means of training an autoencoder to preserve the local geometry of the original data distribution. With that goal in mind, we formulate a **neighborhood reconstruction loss** $\mathcal{L}$ as follows:

$$\mathcal{L}(\theta, \phi; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \frac{1}{|\mathcal{N}(x)|} \sum_{x' \in \mathcal{N}(x)} K(x', x) \cdot \|x' - \tilde{F}_{\theta,\phi}(x'; x)\|^2, \tag{3.2.3}$$

where $K(x', x)$ is a positive symmetric kernel function that determines the weight for each $x' \in \mathcal{N}(x)$. Figure 3.3 illustrates how the neighborhood reconstruction loss can differentiate among the quality of the learned manifolds whose point reconstruction losses are all the same (close to zero): Case 3 has the smallest neighborhood reconstruction loss compared to Case 1 (wrong local geometry) and Case 2 (overfitting). NRAE converges to the vanilla AE – that is, the neighborhood reconstruction loss recovers the point reconstruction loss – if one of the following conditions is met: (i) $\mathcal{N}(x) = \{x\}$, (ii) $K(x', x) = \delta(x', x)$, or (iii) f_θ is linear.

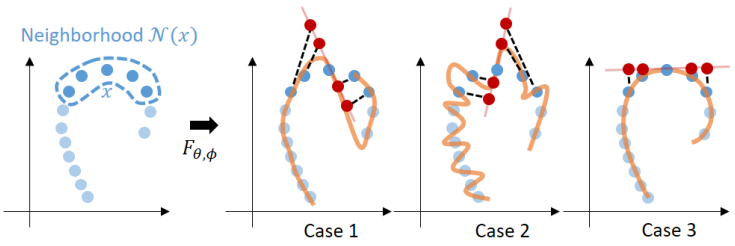


Figure 3.3: The orange curves represent the learned manifolds, the red points represent the neighborhood reconstruction, and the lengths of the black dotted lines represent the neighborhood reconstruction loss.

It is reasonable to ask whether there are any advantages to using an approximation for both the encoder and decoder, i.e., to use a local quadratic (or linear) approximation for the composition map $F_{\theta,\phi}$ rather than just the decoder. As verified below in Section 3.4.3, using a quadratic approximation for both the encoder and decoder results in minimal to no performance improvement (the results for this Extended NRAE (E-NRAE) case are nearly identical to those obtained for NRAE), but the computational requirements increase substantially. As intuition suggests, applying a quadratic approximation for the decoder is sufficient in evaluating the neighborhood reconstruction loss.

3.2.2 Algorithmic Details

Graph construction. The problem of inferring the geometric structure of a data distribution is typically posed as a graph construction problem [24]. We use one of the simplest graph construction methods, the k-NN graph with the Euclidean distance metric. The robustness of our algorithm with respect to the choice of k is tested in the Appendix A.

Kernel design. We choose the following simple kernel

$$K(x', x) = \lambda + (1 - \lambda)\, \delta(x', x), \tag{3.2.4}$$

where $0 \leq \lambda < 1$ and $\delta(x', x) = 1$ if $x' = x$ and zero otherwise. This assigns the weight 1 for the center $x \in \mathcal{N}(x)$ and the weight λ for the remaining neighborhood points.

Batch sampling. To estimate the gradient of the proposed loss function, we use batch sampling for both summations over $\mathcal{D}$ and $\mathcal{N}(x)$. Given a batch $\mathcal{B} \subset \mathcal{D}$, we again sample a batch $\mathcal{B}_x$ from $\mathcal{N}(x)$. We empirically find that forcing each batch $\mathcal{B}_x$ to include x improves convergence. In this thesis, we set $\mathcal{B}_x = \{x, x_n\}$ where x_n is uniformly sampled from $\mathcal{N}(x) - \{x\}$.

3.3 Related Work: Regularization of Autoencoders

In this section, we review some standard autoencoder regularization techniques and their relation to NRAE: i) regularizing latent space distributions, ii) Jacobian regularization, and iii) regularization using neighborhood graphs.

Regularizing latent space distributions. One popular regularization strategy is to enforce the latent space distribution to be close to some user-specified prior distribution; some examples include the Variational Autoencoder (VAE) [25], Adversarial Autoencoder (AAE) [26], and Wasserstein Autoencoder (WAE) [27]. The Gaussian distribution is a popular choice for the prior, but this choice often leads to over-regularization. Several works improve the VAE by learning more complex priors, or an optimal prior in terms of maximizing the training objective function of the VAE [28, 29, 30, 31]. These techniques are developed to make latent space distributions close to some easy-to-sample prior distribution, but are not designed to resolve the two main issues (local geometry and overfitting) addressed in this thesis.

Jacobian regularization. The Contractive Autoencoder (CAE) attempts to enhance robustness of representation by penalizing the Jacobian norm of the encoder function [32]. However, like other encoder-only regularization methods, it often learns a manifold that overfits the data. More recently [33] regularizes the Jacobian of the decoder function to learn a flat manifold. The trained decoder function may produce a smooth manifold but with the wrong local geometry. The main difference between these methods and NRAE is the use of a neighborhood graph containing local connectivity information.

Regularization using neighborhood graphs. Several recent works have developed graph-based autoencoder regularization methods that, at least implicitly, try to learn manifolds with the right local geometry: Generalized Autoencoder (GAE) [34], Topological Autoencoder (TopoAE) [19], Witness Autoencoder (W-AE) [20], Geometry Regularized Autoencoder (GRAE) [21], and Structure-Preserving Variational Autoencoder

(Sp-VAE) [22]. These works focus almost entirely on the encoder when regularizing the latent space distribution (with the exception of GAE [34]); using the regularization term that does not depend on the decoder lead to noise sensitivity and overfitting issues as described in detail above. These methods are also computationally intensive, especially for large scale high-dimensional problems, as a result of expensive pre-processing [21, 22] or computation in training [19, 20].

In contrast, by focusing more on the geometry of the decoded manifold, NRAE is able to learn smooth manifolds with the correct local geometry, simpler to implement, and more scalable since it requires only a single prior construction of the graph without other additional pre-processing steps. Indeed, with the exception of Sp-VAE [22], all other approaches test their algorithms with two-dimensional latent spaces, while we test NRAE for latent spaces with up to 128 dimensions.

For general regularization methods that focus on latent space distributions, we surmise that using our neighborhood reconstruction loss in lieu of the point reconstruction loss may lead to several performance improvements; this is verified in Section 3.4.3, where we combine our neighborhood reconstruction loss with some existing encoder regularization methods and compare their performance against the original methods.

3.4 Experiments

In this section, through extensive experiments with both synthetic and real-world image data, we compare NRAE with a range of existing regularization methods: the Variational Autoencoder (VAE) [25], Wasserstein Autoencoder (WAE) [27], Denoising Autoencoder (DAE) [18], Contractive Autoencoder (CAE) [32], Geometry Regularized Autoencoder (GRAE) [21], and Structure-Preserving Variational Autoencoder (Sp-VAE) [22]. For comparison with Sp-VAE, we augment the regularization term introduced in [22] to a vanilla autoencoder rather than the variational autoencoder – we refer to this autoencoder as the Structure-Preserving Autoencoder (SPAE) – since a direct comparison with the variational autoencoder is already made, and our intent is to examine the effects of the structure-preserving regularization term.

For NRAE, we use both the local quadratic and linear approximations, respectively denoted NRAE-Q and NRAE-L. Our focus is on comparing the manifold smoothing property, geometry preserving property, and generalization ability of NRAE against baseline methods. We refer the reader to the Appendix A for a description of the network architectures used in the experiments, together with implementation details including the hyperparameter tuning strategy.

3.4.1 Manifold Smoothing Property

Smoothness of learned manifolds. We first consider a one-dimensional manifold embedded in $\mathbb{R}^2$, $\{x, \sin(x) | x \in [-\pi, \pi]\}$. Given 50 randomly sampled data points, we add isotropic Gaussian noise with a standard deviation of 0.2. We train the NRAE and baseline AEs with one-dimensional latent spaces. Figure 3.4 and Table 3.1 show that NRAE-L and NRAE-Q successfully re-produce smooth manifolds compared to other baseline methods.

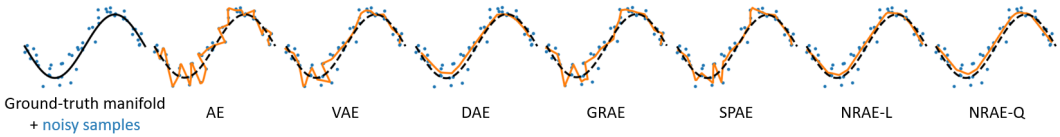


Figure 3.4: The noisy samples (blue) and learned manifolds (orange).

Table 3.1: The mean-squared reconstruction errors of 10,000 clean test data. The averages and standard deviations are computed over ten times run with different noises, and multiplied by 100 (the highest and lowest scores are ignored). The best and second-best results are colored red and blue, respectively.

AE	VAE	WAE	DAE	CAE
1.90 ± 0.23	1.45 ± 0.39	2.02 ± 0.76	1.09 ± 0.21	1.33 ± 0.30
GRAE	**SPAE**	**NRAE-L**	**NRAE-Q**	
1.56 ± 0.43	1.28 ± 0.30	0.29 ± 0.06	0.30 ± 0.07	

Second, we rotate 100 MNIST images of the digit eight 100 times by 3.6 degrees, obtaining a set of 10,000 training data. We train the NRAE and baseline AEs with two-dimensional latent spaces, normalize each latent space, and generate rotating images by sampling regular grids of the latent spaces. NRAE-L and NRAE-Q generate smoothly varying images compared to other baselines (Figure 3.5). The DAE does not generate smoothly varying samples because the noise statistics (Gaussian noise) used in training is different from the statistical noise in the rotating image data.

Finally, to confirm the denoising effect of NRAE on common image data, we train the NRAE and baseline AEs on MNIST and CIFAR10 data corrupted with various levels of Gaussian noise (we use sufficiently large epochs for convergence). We then compare denoising performance measured by the Peak Signal-to-Noise Ratio (PSNR). As shown in Table 3.2, NRAE-L and NRAE-Q outperform other baselines including DAE which performed the third best (the noise statistics used in training is the same as the added noise statistics).

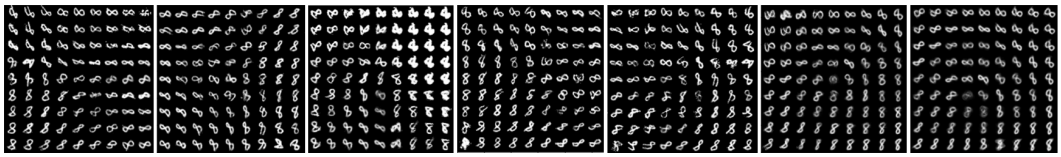


Figure 3.5: Generated samples of rotating digit 8 from regular grids of the latent spaces.

Table 3.2: Comparison of PSNR (higher-the-better). The latent space dimensions are 16 for MNIST and 128 for CIFAR10. The best and second-best results are colored red and blue, respectively.

Dataset	Noise	AE	VAE	WAE	DAE	CAE	GRAE	SPAE	NRAE-L	NRAE-Q
MNIST	0.1	20.19	19.61	19.83	20.69	20.08	20.06	20.01	21.07	21.27
	0.2	17.15	16.36	16.96	18.37	17.24	17.34	17.34	18.55	18.90
CIFAR10	0.1	18.22	19.87	19.54	20.42	20.73	19.79	19.43	22.21	22.27
	0.2	17.84	18.25	17.38	19.30	18.86	16.98	16.90	21.04	20.94

Curvature of learned manifolds. We train the NRAE and baseline AEs on MNIST images with two-dimensional latent spaces, normalize each latent space, then visualize the scalar curvature field (i.e., twice the Gaussian curvature [35]) (Figure 3.6). As shown in Figure 3.6, NRAE-L and NRAE-Q both learn flatter manifolds compared to our baseline AEs.

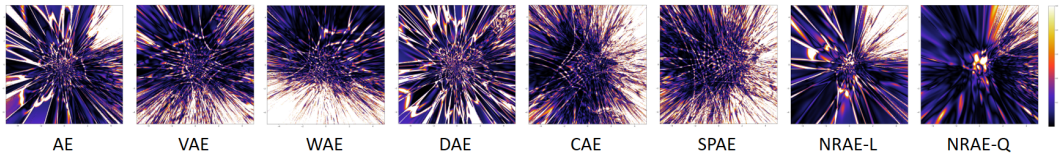


Figure 3.6: The scalar curvature field (brighter-the-larger).

3.4.2 Geometry Preserving Property.

Swiss roll. Consider a one-dimensional Swiss roll $r(\theta) = 0.1 + 0.9 \cdot \theta/(2\pi)$ for $\theta \in [0, 2\pi]$. Given 30 randomly sampled data points, we add isotropic Gaussian noise with a standard deviation of 0.01. We then train the NRAE and baseline AEs with one-dimensional latent spaces. Figure 3.7 shows the reconstruction results for 1000 test data points. Only GRAE, SPAE, and NRAE-Q successfully learn manifolds with the

right local geometry. However, unlike NRAE-Q, the regularization terms in GRAE and SPAE have the effect of increasing reconstruction errors (observe the red rectangles in Figure 3.7). NRAE-L fails to reconstruct points around the point x_r circled in red: the nearest neighbor to x_r in the training set, marked by the blue circle, lies close to the tangent space of the decoded manifold at x_r, i.e., the neighborhood reconstruction loss is small.

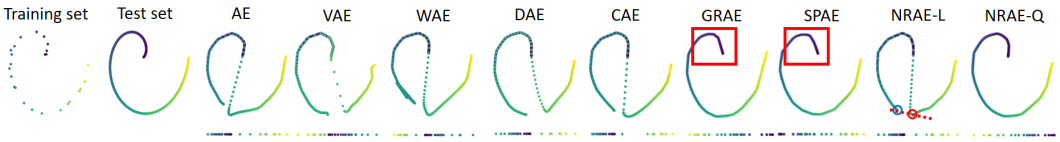


Figure 3.7: Swiss roll data trained with one-dimensional latent spaces. The dots below the figures represent the one-dimensional latent space encoding of the training set.

Rotated/Shifted MNIST. First, as shown in the first row of the two figures in Figure 3.8, we generate two sets of 20 training data: the rotated MNIST images of the digit 3 and shifted MNIST images of the digit 7. We train NRAE and baseline AEs with one-dimensional latent spaces, encode the training data to the latent spaces, sort the encoded values in ascending order, and decode them to generate the images (Figure 3.8). Only SPAE, NRAE-L, and NRAE-Q successfully learn manifolds of the rotated/shifted images with the correct geometry.

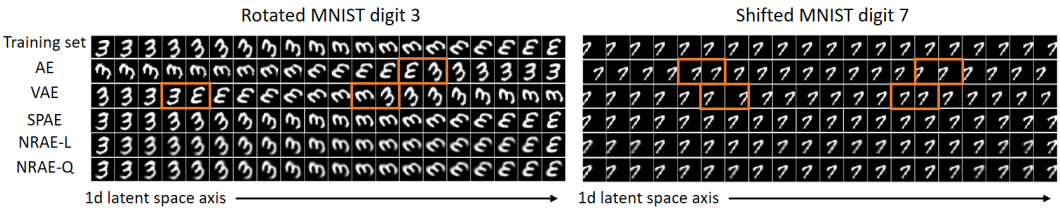


Figure 3.8: Generated rotated/shifted MNIST images. Discontinuities are marked by orange boxes.

Second, we rotate an MNIST image of the digit six 300 times by 1.2 degrees and obtain 300 training data; 40 of these are visualized in the first row of Figure 3.9. We further generate 1000 test data in a similar manner. We then train the AE, DAE, CAE, and NRAE with one-dimensional circular latent spaces by adding one layer to the end of the encoder corresponding to $z \to z/\|z\|$, and visualize which data are encoded to

which point in the latent spaces via the cyclic color map. Only NRAE-Q is able to learn a manifold with the correct local geometry.

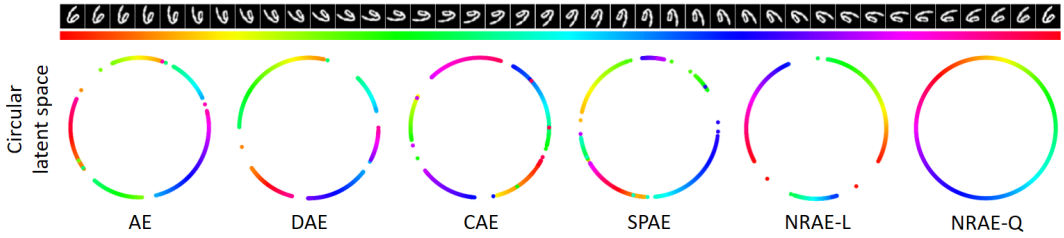


Figure 3.9: Circular latent space encoding of the rotated MNIST images of digit 6.

3.4.3 Generalization

Test data reconstruction. With various large-scale standard benchmark image data, we show that NRAE generalizes better to the test data compared to other baseline methods. We use the test reconstruction Mean Square Error (MSE) for quantitative comparison.

First, we train fully-connected neural networks with the MNIST, FMNIST, and KMNIST data by varying the number of training data from $1000, 2000 \ldots, 10,000$, and compare the test reconstruction MSEs. The numbers for the validation and test data are fixed at $10,000$ and $50,000$, respectively. Figure 3.10 shows the MSEs as a function of the number of training data. The MSEs decrease as the number of training data increases, while those of the NRAE are lower than most of the other baselines regardless of the number of training data.

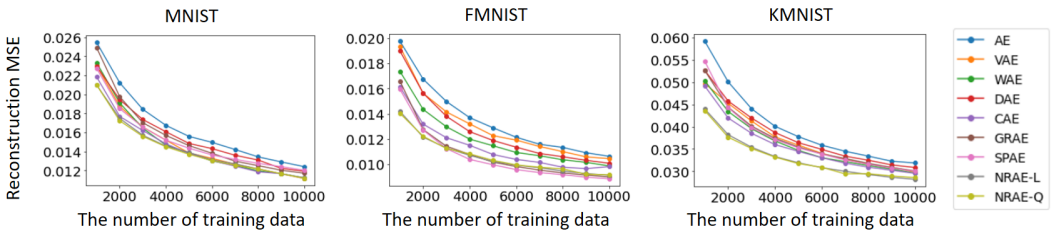


Figure 3.10: The test reconstruction MSEs as the number of training data changes.

Second, we train convolutional neural networks and compare the test reconstruction MSEs (Table 3.3). We conduct two experiments i) with the entire public data denoted by L (large) and ii) with the subset of data denoted by S (small). The number

of selected subsets are around $20 \sim 30\%$ of the entire data. NRAE-L and NRAE-Q produce lower reconstruction MSEs than most of the other baselines, meaning that the manifolds learned with our methods better generalize to the test data. The CAE experiment on CELEBA data is excluded because of its high computational cost for high-dimensional data. Other measures such as the Frechet-Inception Distance (FID) scores [36] and the Evidence Lower Bound (ELBO) are reported in the Appendix A.

Table 3.3: The test reconstruction MSEs, the lower the better. The latent space dimensions are 16, 32, 32, 32, 64, 128, 128, 128 for MNIST, FMNIST, KMNIST, Omniglot, SVHN, CIFAR10, CIFAR100, and CELEBA, respectively. The best and second-best results are colored red and blue, respectively.

Dataset	Size	AE	VAE	WAE	DAE	CAE	GRAE	SPAE	NRAE-L	NRAE-Q
MNIST	S	0.01002	0.01091	0.01009	0.00999	0.00998	0.01004	0.00989	0.00953	0.00968
	L	0.00688	0.00756	0.00690	0.00684	0.00692	0.00696	0.00694	0.00649	0.00683
FMNIST	S	0.01485	0.01652	0.01428	0.01446	0.01319	0.01331	0.01363	0.01289	0.01277
	L	0.01118	0.01235	0.01106	0.01099	0.01052	0.01060	0.01065	0.01060	0.01044
KMNIST	S	0.03267	0.03234	0.03283	0.03280	0.03279	0.03206	0.03268	0.03071	0.03021
	L	0.02844	0.02963	0.02776	0.02814	0.02762	0.02753	0.02732	0.02564	0.02602
Omniglot	S	0.03038	0.03627	0.03078	0.03068	0.02714	0.02967	0.02889	0.02668	0.02631
	L	0.02704	0.03192	0.02728	0.02696	0.02567	0.02648	0.02644	0.02578	0.02539
SVHN	S	0.00320	0.00420	0.00320	0.00369	0.00273	0.00317	0.00307	0.00202	0.00192
	L	0.00174	0.00204	0.00190	0.00177	0.00178	0.00173	0.00175	0.00148	0.00147
CIFAR10	S	0.01466	0.01620	0.01431	0.01427	0.01208	0.01452	0.01504	0.00768	0.00691
	L	0.00960	0.01123	0.00863	0.00900	0.00755	0.00832	0.00898	0.00629	0.00587
CIFAR100	S	0.01465	0.01713	0.01463	0.01484	0.01369	0.01391	0.01477	0.00765	0.00717
	L	0.01015	0.01064	0.00951	0.00862	0.00842	0.00910	0.00912	0.00678	0.00635
CELEBA	S	0.00780	0.00937	0.00830	0.00782	-	0.00814	0.00861	0.00608	0.00747
	L	0.00613	0.00646	0.00630	0.00590	-	0.00595	0.00665	0.00563	0.00565

Compatibility with other regularization methods. In this section, we show that the NRAE loss function can be used together with other existing latent space distribution regularization methods to improve their generalization performance. We train convolutional neural networks with the MNIST and CIFAR10 data (small). As shown in Table 3.4, the NRAE loss improves WAE, CAE, and SPAE by significant margins. For VAE, there is only little improvement, primarily because the VAE regularization term is too dominant in training. Interestingly, some cases of WAE, CAE, and SPAE combined with the NRAE loss show even better generalization performance compared to NRAE alone. These results imply that a proper combination of the neighborhood reconstruction loss and existing regularization methods can lead to further improvements in performance.

Table 3.4: Comparison of the test reconstruction MSEs for VAE, WAE, CAE, GRAE, and SPAE before and after being combined with the NRAE loss function. MNIST and CIFAR10 data (small) are used.

Dataset	NRAE-L	NRAE-Q	VAE	NRVAE-L	NRVAE-Q	WAE	NRWAE-L	NRWAE-Q
MNIST	0.00953	0.00968	0.01091	0.01089	0.01098	0.01009	0.00952	0.00966
CIFAR10	0.00768	0.00691	0.01620	0.01613	0.01609	0.01431	0.00707	0.00684
	NRAE-L	NRAE-Q	CAE	NRCAE-L	NRCAE-Q	SPAE	NRSPAE-L	NRSPAE-Q
MNIST	0.00953	0.00968	0.00998	0.00936	0.00965	0.00989	0.00942	0.00975
CIFAR10	0.00768	0.00691	0.01208	0.00718	0.00724	0.01504	0.00723	0.00703
	NRAE-L	NRAE-Q	GRAE	NRGRAE-L	NRGRAE-Q			
MNIST	0.00953	0.00968	0.01004	0.00945	0.00974			
CIFAR10	0.00768	0.00691	0.01452	0.00716	0.00694			

Extended NRAE. Instead of using a quadratic (or linear) approximation of f_θ, we can approximate the composition function $F_{\theta,\phi} = f_\theta \circ g_\phi$ to define the neighborhood reconstruction loss. While NRAE only requires the computation of the Jacobian-vector product or Hessian-vectors product of f_ϕ, E-NRAE requires the computation of these quantities for $F_{\theta,\phi}$. We train convolutional neural networks on the MNIST and CIFAR10 data (small) and compare their generalization abilities and computational requirements. As shown in Table 3.5, the extended versions show comparable generalization performance to the original version, yet take a longer per-epoch runtime (100 batch size and 10,000 training data).

Table 3.5: Comparison of the test reconstruction MSEs and per-epoch runtime estimates for NRAE-L, NRAE-Q, E-NRAE-L, and E-NRAE-Q using MNIST and CIFAR10 data (small).

Dataset	Metric	NRAE-L	E-NRAE-L	NRAE-Q	E-NRAE-Q
MNIST	mse	0.00953	0.01000	0.00968	0.01017
	runtime	21.57 s	24.85 s	34.73 s	40.96 s
CIFAR10	mse	0.00768	0.00712	0.00691	0.00732
	runtime	59.14 s	61.28 s	89.24 s	100.18 s

3.5 Conclusion

We have proposed a new graph-based autoencoder, the Neighborhood Reconstructing Autoencoder (NRAE), that is capable of learning accurate manifolds that are robust to noisy training data and have the correct local connectivity and geometry, while being easy to implement, scalable, and computationally efficient. Neighborhood graphs that

capture the local geometry of the data distribution are combined with local quadratic (or linear) approximations of the decoder function to formulate a new neighborhood reconstruction loss, which turns out to be a generalization of the original point reconstruction loss. Through extensive experiments with both synthetic and standard image datasets, we have demonstrated the manifold smoothing property, geometry preserving property, and the generalization performance advantages – in some cases by significant margins – of our method.

Further, we have empirically verified that (i) a proper combination of the neighborhood reconstruction loss and existing regularization terms that focus on the latent space distributions can lead to further improvements in performance and (ii) in neighborhood reconstruction loss, approximating the decoder only is sufficient for learning the correct manifold while being computationally more efficient than approximating both the encoder and decoder.

Our algorithm can be further enhanced in a number of different ways. First, the current implementation of NRAE uses the k-NN graph construction with Euclidean distance metric and a simple kernel function that outputs binary values. These choices are made for simplicity, and clearly sub-optimal. Other combinations of the graph construction method (e.g., the persistent homology [37]), distance metric (e.g., using deep metric learning), and kernel function (e.g., the Gaussian kernel) are worth exploring. Second, NRAE can be extended to a stochastic model in a number of ways, e.g., using the probabilistic autoencoder [38] that uses the latent variable model, or the energy-based approach as done in [39].

4
Minimum Curvature Manifold Learning

4.1 Introduction

Autoencoders are widely used to identify, given a set of high-dimensional data, the underlying lower-dimensional manifold structure and its coordinate space, simultaneously [9]. The decoder explicitly parameterizes the data manifold as a mapping from a lower-dimensional coordinate space (i.e., latent space) to the high-dimensional data space, and the encoder maps data points to their corresponding coordinates (i.e., latent values). However, vanilla autoencoders trained to reconstruct the given training data often learn manifolds that severely overfit to noisy training data or are wrong in regions where there are fewer data, impairing their manifold learning performances.

It has been recently discovered by [13] that autoencoder regularization methods that focus on regularizing the latent space distributions determined entirely by the encoders [25, 40, 26, 32] are not sufficient to learn correct manifolds, yet it is important to properly regularize the decoders that parameterize the manifolds. In [13], neighborhood graphs constructed from data are successfully utilized to regularize the local geometry and connectivity of the manifold, significantly improving the manifold learning accuracy. However, the underlying premise behind this method is that the graph has to be accurate, yet constructing a correct graph may not be always straightforward.

There are some graph-free methods such as the denoising autoencoder [18] and reconstruction contractive autoencoder [41] that regularize not only an encoder but also a decoder. They can learn manifolds that are robust to noise to some extent, but when the noise level is large, the performance is often less-than-desirable, and they do not always produce correct manifolds, especially in regions where there are fewer data (discussed in more detail in Section 4.4.2).

Since the decoder needs to be regularized, one may come up with some naive

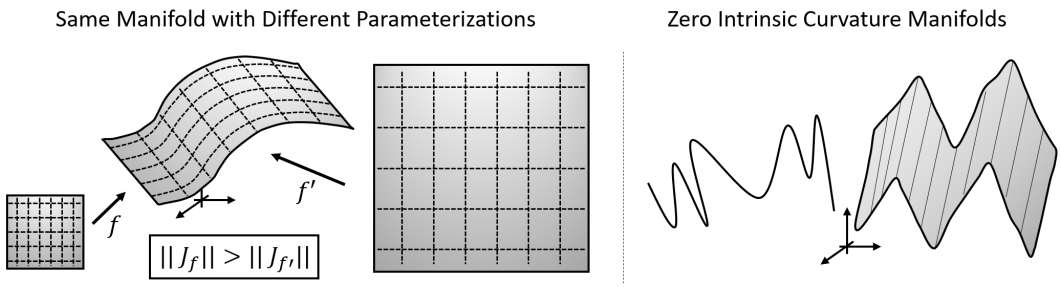


Figure 4.1: *Left*: Two decoders f and f' parameterize the same data manifold where the norm of Jacobian of f' is smaller than that of f, i.e., $\|J_f\| > \|J_{f'}\|$. *Right*: A curve and developable surface embedded in $\mathbb{R}^3$ have zero intrinsic curvatures.

regularization strategies such as minimizing the norm of the decoder's Jacobian or Hessian, considering them as measures of the manifold's smoothness. However, these norms do not properly capture any geometric quantity of the manifold because they are not reparametrization-invariant (or coordinate-invariant). As shown in Figure 4.1 (*Left*), just by increasing the volume of the latent space without actually changing the manifold, i.e., re-parametrizing the manifold $f \mapsto f'$, the above norms can be minimized.

Just recently, a coordinate-invariant geometric distortion measure has been introduced to regularize the decoder to be a geometry-preserving mapping, which is called the isometric regularization [14], so that the data space geometry is preserved in the latent space. Minimizing this distortion measure implicitly forces the learned manifold to have zero intrinsic curvature – which only depends on distances measured within the manifold (e.g., a cylinder's side surface has zero intrinsic curvature unlike the spherical surface) –, resulting in some geometrically meaningful manifold regularization effects.

The intrinsic curvature, however, does not capture how the manifold lies in the data space,[1] and thus minimizing the manifold's intrinsic curvature may not be enough to learn correct manifolds. For example, curves and developable surfaces[2] in $\mathbb{R}^3$ always have zero intrinsic curvatures, e.g., Figure 4.1 (*Right*), regardless of how severely they are curved from an extrinsic point of view [35].

The main contribution is a *coordinate-invariant extrinsic curvature minimization framework* for autoencoder regularization, which we refer to a **Minimum Curvature**

[1] Manifold's intrinsic properties are defined without involving any embedding.

[2] A developable surface can be formed by bending or rolling a planar surface without stretching or tearing.

Autoencoder (MCAE), that is graph-free and effectively improves the manifold learning accuracy given a noisy or small training dataset. Specifically, we develop a coordinate-invariant extrinsic curvature measure of the learned manifold, by investigating how smoothly tangent space changes on the manifold, and use it as a regularization term.

To make things more explicit, let $\mathcal{M}$ be a manifold of dimension m embedded in $\mathbb{R}^D$. Consider a mapping T that maps a point x in $\mathcal{M}$ to its tangent space $T_{\mathsf{x}}\mathcal{M}$, a linear subspace that has the dimension of m attached at x, i.e., $T(\mathsf{x}) = T_{\mathsf{x}}\mathcal{M}$. The set of all linear subspaces of dimension m in $\mathbb{R}^D$ forms a manifold called the Grassmann manifold denoted by $\mathsf{Gr}(m, \mathbb{R}^D)$ [42], and thus the mapping T can be viewed as a mapping between two Riemannian manifolds, i.e., $T : \mathcal{M} \to \mathsf{Gr}(m, \mathbb{R}^D)$. By using the Dirichlet energy [43], a natural smoothness measure of mappings between two Riemannian manifolds defined in a coordinate-invariant way, we formulate an extrinsic curvature measure. We also propose a practical estimation strategy of the curvature measure that can be used for high-dimensional problems, reducing computation costs.

Experiments on diverse image and motion capture data confirm that, compared to existing graph-free regularized autoencoders, our MCAE improves manifold learning accuracy for noisy and small training datasets. In particular, our experiments show that even compared to the methods specially designed to be robust to input perturbations such as the DAE [18] and RCAE [41], the MCAE shows comparable or even in some cases significantly higher robust manifold learning performance.

4.2 Geometric Preliminaries

4.2.1 Grassmann Manifold

In this section, we review the Grassmann manifold and its Riemannian geometry from a matrix-analytic perspective. The Grassmann manifold is defined as the set of all m dimensional linear subspaces of the Euclidean space $\mathbb{R}^D$, denoted by $\mathsf{Gr}(m, \mathbb{R}^D)$; this can be identified with the set of orthogonal rank-m projection matrices as follows:

$$\mathsf{Gr}(m, \mathbb{R}^D) = \{P \in \mathbb{R}^{D\times D} \mid P^T = P,\ P^2 = P,\ \mathsf{rank}(P) = m\}, \tag{4.2.1}$$

which is an $m(D-m)$ dimensional manifold; which associates $P \in \mathsf{Gr}(m, \mathbb{R}^D)$ with the linear subspace $\mathsf{range}(P) \subset \mathbb{R}^D$. This is an implicit parametrization of the Grassmann manifold considered as being embedded in the Euclidean space $\mathbb{R}^{D\times D}$. For more formal and detailed descriptions of the Grassmann manifold, we refer to [42].

Given a rank-m matrix $J \in \mathbb{R}^{D\times m}$, one may want to consider its range, an m-dimensional linear subspace in $\mathbb{R}^D$, as an element of the Grassmann manifold. The

embedding $E : \mathbb{R}^{D\times m} \to \mathsf{Gr}(m, \mathbb{R}^D)$ such that $E(J) = J(J^TJ)^{-1}J^T$ properly converts J to the element of (4.2.1). We note that (i) range$(J) =$ range$(E(J))$ and (ii) $E(J) = E(JA)$ for any $m \times m$ invertible matrix $A \in \mathbb{R}^{m\times m}$ since the transformation $J \mapsto JA$ does not change the range.

Next, we introduce the basic Riemannian structure of the Grassmann manifold. At a point $P \in \mathsf{Gr}(m, \mathbb{R}^D)$, the tangent space is defined as follows:

$$T_P\mathsf{Gr}(m, \mathbb{R}^D) := \{V \in \mathbb{R}^{D\times D} \mid V^T = V,\ VP + PV = V\}, \tag{4.2.2}$$

which can be derived from (4.2.1) by differentiating the constraints. One canonical choice of the Riemannian metric is given as follows:

$$\langle V_1, V_2\rangle := \frac{1}{\sqrt{2}}\mathsf{Tr}(V_1^TV_2) \quad \text{for} \quad V_1, V_2 \in T_P\mathsf{Gr}(m, \mathbb{R}^D). \tag{4.2.3}$$

This metric is invariant under the orthogonal transformation, i.e., $\langle V_1, V_2\rangle = \langle RV_1, RV_2\rangle$ for any $D \times D$ orthogonal matrix R.

4.2.2 Dirichlet Energy for Mappings between Riemannian Manifolds

This section introduces the Dirichlet energy for mappings between two Riemannian manifolds. Let $\mathcal{M}$ and $\mathcal{N}$ be Riemannian manifolds of dimension m and n; we will consider a differentiable mapping f : $\mathcal{M} \to \mathcal{N}$. We will assume x $\in \mathcal{M}$ is explicitly parametrized by local coordinates as $x \in \mathbb{R}^m$ and the Riemannian metric at x $\in \mathcal{M}$ is expressed as $m \times m$ positive-definite matrix $G(x) = (g_{ij}(x)) \in \mathbb{R}^{m\times m}$, and $\mathcal{N}$ is embedded in the Euclidean space of higher dimension as $N \subset \mathbb{R}^d$ $(d \gg n)$ and the Riemannian metric at y $\in \mathcal{N}$ is given as $\langle\cdot,\cdot\rangle_y$ for $y \in N$ (e.g. Grassmann manifold). The mapping f is expressed as $f : \mathbb{R}^m \to N \subset \mathbb{R}^d$ such that $y = f(x)$.

The Dirichlet energy, a global measure of how much the mapping f changes, is defined as follows:

$$\int_{\mathcal{M}} \sum_{i=1}^{m}\sum_{j=1}^{m} g^{ij}(x)\langle\frac{\partial f}{\partial x^i}(x), \frac{\partial f}{\partial x^j}(x)\rangle_{f(x)} \sqrt{\det\ G(x)}\ dx^1 \cdots dx^m, \tag{4.2.4}$$

where $g^{ij}(x)$ denotes (i, j)-th element of the inverse of $G(x)$ and $\sqrt{\det\ G(x)}\ dx^1 \cdots dx^m$ is the Riemannian volume form, which corresponds to the integral functional from the theory of harmonic maps; this integral is an intrinsic quantity (i.e., coordinate-invariant). We note that the integrand is a local measure of how much the mapping f changes. We refer to the extensive literature on the theory and applications of harmonic maps, e.g., [43, 44, 45, 46, 14].

4.3 Minimum Curvature Autoencoders

In this section, we propose a regularized autoencoder based on the principle of minimum curvature manifold learning. Throughout, we consider a data space $\mathbb{R}^D$ and latent space $\mathbb{R}^m$ $(D \gg m)$ and denote a parametric encoder by $g_\phi : \mathbb{R}^D \to \mathbb{R}^m$ such that $z = g_\phi(x)$, and a parametric decoder by $f_\theta : \mathbb{R}^m \to \mathbb{R}^D$ such that $x = f_\theta(z)$. The manifold parametrized by the decoder will be denoted by $\mathcal{M}_\theta$, and the Jacobian of the decoder by $J_\theta(z) = \frac{\partial f_\theta}{\partial z}(z)$. Given a set of data points $\{x_i \in \mathbb{R}^D\}_{i=1}^N$, the empirical data distribution will be denoted by $\hat{p}(x) := \frac{1}{N}\sum_{i=1}^N \delta(x - x_i)$ and the latent space distribution encoded by g_ϕ by $\hat{p}_\phi(z) := \frac{1}{N}\sum_{i=1}^N \delta(z - g_\phi(x_i))$. The subscripts show what variables each function or geometric object depends on, either θ or ϕ.

4.3.1 Coordinate-Invariant Extrinsic Curvature Measure

In this section, we formulate a coordinate-invariant (i.e., reparametrization-invariant) extrinsic curvature measure of the manifold $\mathcal{M}_\theta$ embedded in $\mathbb{R}^D$. We begin by introducing the notion of coordinate-invariance:

Definition 4.3.1. Given a manifold $\mathcal{M}$ of dimension m embedded in $\mathbb{R}^D$, let $f : \mathbb{R}^m \to \mathcal{M}$ be its explicit parametrization. A functional $\mathcal{F}(f)$ is coordinate-invariant (i.e., reparametrization-invariant) if, given any invertible mapping or coordinate transformation (i.e., reparametrization) $h : \mathbb{R}^m \to \mathbb{R}^m$, $\mathcal{F}(f) = \mathcal{F}(f \circ h^{-1})$.

The coordinate-invariance is necessary to properly measure any geometrically meaningful quantity of the manifold. For example, the integration of the Frobenius norm of J_θ in coordinate space $\mathbb{R}^m$ is not coordinate-invariant, and hence does not capture any geometrically meaningful quantity of $\mathcal{M}_\theta$.

Now, we define a coordinate-invariant extrinsic curvature measure of $\mathcal{M}_\theta$. The core idea is to define a local measure of the extrinsic curvature by measuring how fast the tangent space $T_x\mathcal{M}_\theta$ changes within the neighborhood of x, and then integrate it over the manifold to define a global curvature measure. For this purpose, let a pair of mappings, encoder g_ϕ and decoder f_θ, be a coordinate system for $\mathcal{M}_\theta$, and consider a mapping $T : \mathbb{R}^m \to \mathsf{Gr}(m, \mathbb{R}^D)$ such that $T(z)$ is the element of the Grassmann manifold (4.2.1) whose range is equal to $T_{f_\theta(z)}\mathcal{M}_\theta$. We note that the range of the Jacobian matrix $J_\theta(z) \in \mathbb{R}^{D\times m}$ is $T_x\mathcal{M}_\theta$, hence, by using the embedding $E : \mathbb{R}^{D\times m} \to \mathsf{Gr}(m, \mathbb{R}^D)$ such that $E(J_\theta) := J_\theta(J_\theta^T J_\theta)^{-1} J_\theta^T$, we can explicitly write the mapping T as $T(z) = E(J_\theta(z))$.

Let $\mathcal{M}_\theta$ be assigned with the Riemannian metric induced from the data space Riemannian metric $H(x)$, so that the metric expressed in the coordinate space is

$J_\theta^T(z)H(f_\theta(z))J_\theta(z)$, and $\mathsf{Gr}(m, \mathbb{R}^D)$ be assigned with the Riemannian metric in (4.2.3). We use the dirichlet energy in (4.2.4) of the mapping T as a coordinate-invariant extrinsic curvature measure, where the integral is replaced by the expectation over $\hat{p}_\phi(z)$:

Definition 4.3.2. Given an encoder g_ϕ, decoder f_θ, and empirical distribution in coordinate space $\hat{p}_\phi(z)$, the global extrinsic curvature measure of $\mathcal{M}_\theta$ with respect to $\hat{p}_\phi(z)$ is defined as

$$\mathcal{C}(\theta, \phi) := \mathbb{E}_{z\sim\hat{p}_\phi(z)}[\sum_{i=1}^{m}\sum_{j=1}^{m}(J_\theta^T H J_\theta)_{ij}^{-1}\mathsf{Tr}(\frac{\partial}{\partial z^i}(E(J_\theta))\frac{\partial}{\partial z^j}(E(J_\theta)))]. \tag{4.3.5}$$

In this thesis, we use the identity metric for the ambient data space, i.e., $H(x) = I_{D\times D}$.

Proposition 4.1. *The curvature measure $\mathcal{C}(\theta, \phi)$ in Definition 4.3.2 is coordinate-invariant, i.e., for another pair of encoder $g_{\phi'} := h \circ g_\phi$ and decoder $f_{\theta'} := f_\theta \circ h^{-1}$ with any invertible map or coordinate transformation h such that $z' = h(z)$, the measure is invariant, i.e., $\mathcal{C}(\theta, \phi) = \mathcal{C}(\theta', \phi')$.*

Proof. The proof is given in the Appendix B.2 □

Our definition of the curvature generalizes classical definition of the curvature of a curve embedded in $\mathbb{R}^3$ from differential geometry [47] (please see Appendix B.3 for more details).

With the proposed curvature measure, we define a regularized autoencoder where the loss function consists of the following two terms i) reconstruction error term for manifold learning and ii) regularization term $\mathcal{C}(\theta, \phi)$ for curvature minimization:

$$\min_{\theta,\phi}\ \mathbb{E}_{x\sim\hat{p}(x)}[\|x - f_\theta \circ g_\phi(x)\|^2] + \alpha\ \mathcal{C}(\theta, \phi), \tag{4.3.6}$$

where α is the regularization coefficient, which we refer to as the **Minimum Curvature Autoencoder (MCAE)**.

4.3.2 Practical Implementations

This section introduces two practical strategies for computation of the curvature measure (4.3.5).

Augmented Distribution: In (4.3.5), the local curvature measure is expected over the empirical latent space distribution. However, the influence of the measure is then

limited to regions where data is available; thus the manifold's curvature in regions where data is no data may not be properly regularized. In practice, we use data augmentation to resolve this issue. Following [33, 14], we use the modified mix-up data-augmentation method with a parameter $\eta > 0$, where $\hat{p}_\phi(z)$ is augmented by $z = \delta z_1 + (1-\delta)z_2$ such that $z_i \sim p_\phi(z), i = 1, 2$, where δ is uniformly sampled from $[-\eta, 1+\eta]$. We set $\eta = 0.2$ throughout.

Stochastic Trace Estimation: At first glance, the curvature measure (4.3.5) seems computationally very expensive, because it involves the computation of the full Jacobian J_θ of a deep neural network and derivative of the Jacboaidn $\frac{\partial J_\theta}{\partial z}$, and we even need to backpropagate through them when using the standard stochastic gradient descent algorithms. To efficiently compute the measure in practice, we use the Hutchinson's trace estimator [48], i.e., $\mathsf{Tr}(A) = \mathbb{E}_{v\sim\mathcal{N}(0,I)}[v^T A v]$, then the curvature measure $\mathcal{C}(\theta, \phi)$ has the following expression:

$$\mathcal{C}(\theta,\phi) = \mathbb{E}_{z\sim\hat{p}_\phi(z), v\sim\mathcal{N}(0,I_m), w\sim\mathcal{N}(0,I_D)}[v^T \frac{\partial(w^T E(J_\theta))}{\partial z} \frac{\partial(E(J_\theta)w)}{\partial z} G_\theta^{-1} v], \quad (4.3.7)$$

where I_k is the $k \times k$ identity matrix and $G_\theta = J_\theta^T H J_\theta$. To implement this computationally efficiently, we use the Jacobian-vector and vector-Jacboian products in multiple times: (i) for $E(J_\theta)w = J_\theta G_\theta^{-1} J_\theta^T w$, we first use the vector-Jacobian product for $J_\theta^T w$ and the Jacobian-vector product for $J_\theta(G_\theta^{-1} J_\theta^T w)$, and (ii) for $\frac{\partial(E(J_\theta)w)}{\partial z} v$ and $\frac{\partial(E(J_\theta)w)}{\partial z}(G_\theta^{-1} v)$, we use the Jacobian-vector products. These techniques make the computation of (4.3.5) tractable for high-dimensional complex problems. Surprisingly, for the estimation of (4.3.7), using one sample of v and w at each $z \sim \hat{p}_\phi(z)$ was sufficient to train MCAE in our later experiments. When the latent space is high-dimensional, the matrix inverse computation G_θ^{-1} takes up most of the computation time. Using an approximate inverse can significantly reduce the computation time, see the Appendix B.6.

4.4 Experiments

4.4.1 Parameter Sweep

We first provide an empirical study on the effect of the most important parameter of MCAE, the regularization coefficient α. Intuitively, as α increases, the tendency to minimize the extrinsic curvature of the manifold becomes stronger, so the learned manifold will become closer to a linear subspace. And, if α is too small, the learned

manifold will not be different from that of the vanilla autoencoder; hence it is important to select an appropriate value for α depending on the dataset.

Figure 4.2 shows how α affects the learned manifold in MCAE with two examples. In the upper figure, given noisy two-dimensional data points, we train MCAEs with one-dimensional latent spaces. In the lower figure, given sparse three-dimensional data points constrained on the 2-sphere $S^2 := \{x \in \mathbb{R}^3 \mid \|x\| = 1\}$, we train MCAEs with one-dimensional latent spaces, where the decoder outputs are normalized to be in S^2. As can be seen, $\alpha = 0.01$ and $\alpha = 0.0001$ are good values for the upper and lower examples, respectively. In practice, we can find the optimal value of α with a proper validation criteria (e.g., mean reconstruction error for validation data).

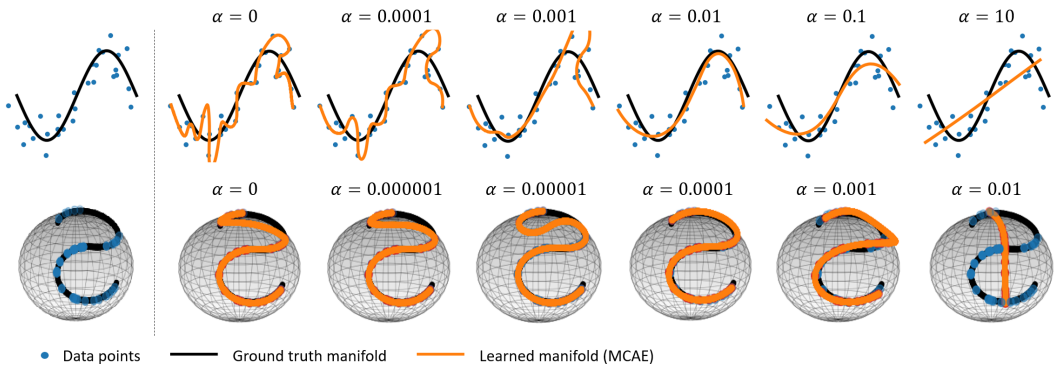


Figure 4.2: Learned manifold becomes flatter as the regularization coefficient α increases. *Upper*: Learned data manifolds of 1d sin-curve and noisy training data points. *Lower*: Learned data manifolds of 1d S-curve projected to the 2-sphere and sparse training data points.

4.4.2 Comparison to Other Regularization Methods

In this section, we compare the proposed MCAE with other regularized autoencoders and highlight the differences. Please refer to Appendix B.1 for more detailed comparisons.

Comparison to Isometrically Regularized Autoencoders: In the Isometrically Regularized Autoencoder (IRAE) [14], the decoder is regularized to be a scaled isometry; similar to (4.3.6), a regularization term that measures how far f_θ from being a scaled isometry is added to the reconstruction error term with the regularization coefficient α. This regularization implicitly forces the learned manifold $\mathcal{M}_\theta$ to have zero

intrinsic curvature, but not the extrinsic curvature; therefore it is at first glance expected that, when learning a one-dimensional manifold, the IRVE should not have any meaningful manifold regularization effect (since one-dimensional manifolds always have zero intrinsic curvatures).

Counterintuitively, as shown in Figure 4.3(a), our experiments show that the extrinsic curvature of the one-dimensional manifold learned by IRAE decreases as α increases. If the decoder's hypothesis space was a set of arbitrary smooth functions, this result would not have been obtained, but since the hypothesis space defined as the set of neural networks is smaller, the isometric regularization seems to reduce the extrinsic curvature at the expense of obtaining the isometric representations. Figure 4.3(b) shows how the reconstruction MSE for clean test data varies as a function of the extrinsic curvature of the learned manifold by IRAE and MCAE. As the curvature decreases or the regularization coefficient increases (from left to right), the test reconstruction MSE decreases, reaches a minimum, and then increases again. We note that the graph of MCAE lies lower than that of IRAE, implying that the MCAE can learn a more accurate manifold than the IRAE.

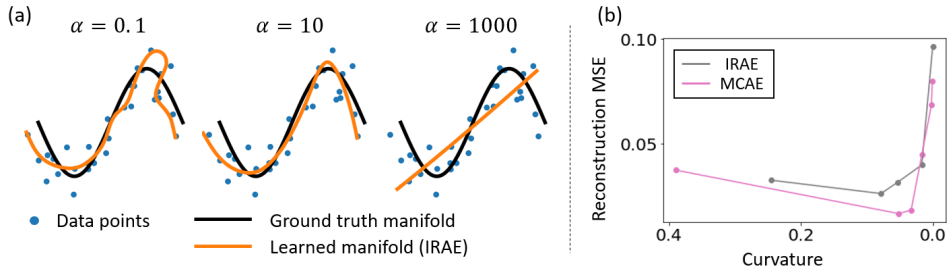


Figure 4.3: (a) Learned manifold by IRAE becomes flatter as the regularization coefficient α increases. (b) Test data reconstruction MSE (i.e., manifold learning accuracy) as a function of the extrinsic curvature obtained by IRAE and MCAE.

Comparison to Denoising and Reconstruction Contractive Autoencoders: Denoising Autoencoder (DAE) [18] and Reconstruction Contractive Autoencoder (RCAE) [41] are intuitive and straightforward regularization methods for learning manifolds robust to input perturbations. As shown in Figure 4.4 (*Upper*), the DAE and RCAE learn manifolds robust to noise to some extent. However, as shown in Figure 4.4 (*Lower*), for the projected S-curve example in Figure 4.2 (*Lower*), they still learn wrong manifolds in regions where there are fewer data and do not improve the vanilla autoencoder. On the other hand, the MCAE explicitly regularizes the learned manifold to have a small curvature globally and improves the manifold learning accuracy.

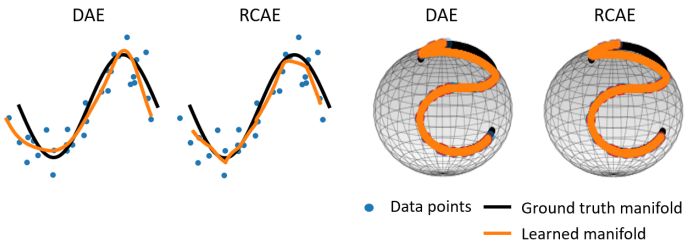


Figure 4.4: Learning by DAE and RCAE for examples in Figure 4.2.

Table 4.1: Averages and standard errors of the test data set reconstruction MSEs (5 times run) for the sincurve example in Figure 4.2 (*Upper*) with various Gaussian noise of standard deviations $0.1, 0.2, 0.3$, the lower the better. The best results are marked in bold. The numbers are written in units of 10^{-3}.

Noise	AE	VAE	DAE	RCAE
0.1	3.98 ± 0.22	2.05 ± 0.36	2.23 ± 0.26	2.95 ± 0.44
0.2	22.7 ± 2.9	6.34 ±0.72	6.99 ± 1.04	12.5 ± 1.1
0.3	68.5 ± 17.3	13.5 ± 2.3	17.8 ± 3.4	30.5 ± 7.9
	DCAE	DHAE	IRAE	MCAE
0.1	2.81 ± 0.21	2.68 ± 0.31	1.63 ± 0.25	**1.28 ± 0.13**
0.2	10.9 ± 1.02	14.9 ± 2.87	6.59 ± 1.09	**4.56 ± 0.69**
0.3	30.7 ± 4.8	46.9 ± 12.5	20.2 ± 5.1	**9.80 ± 1.55**

Quantitative Comparisons of Noise Robustness: As seen from the above examples, besides the proposed MCAE, the IRAE, DAE, RCAE all have the robustness properties to noise. We quantitatively compare the robust manifold learning performance given noisy input training data with the sincurve example in Figure 4.2 (*Upper*) with various noise levels, i.e., Gaussian noise with standard deviations of $0.1, 0.2, 0.3$. In addition to the IRAE, DAE, RCAE, we compare the MCAE with the vanilla Autoencoder (AE) and other regularized autoencoders such as the Variational Autoencoder (VAE) [25], Decoder Contractive Autoencoder (DCAE), and Decoder Hessian Contractive Autoencoder (DHAE), where the DCAE and DHAE minimize the decoder's Jacobian norm and the decoder's Hessian norm, respectively. Table 4.1 shows the averages and standard errors of the test data set reconstruction MSEs, the lower the better. The MCAE produces the lowest errors regardless of the noise level.

4.4.3 Image Data

Grayscale Image: First, we investigate the manifold learning performance of MCAE compared to the other regularized autoencoders with the standard grayscale image data (*MNIST, Fashion-MNIST, KMNIST*) as the number of training + validation data and noise level varies. We use two-layer fully connected neural networks (512 nodes per layer) for both encoder and decoder with ELU activation functions, and the latent space dimensions are $16, 32, 32$, respectively.

Figure 4.5 shows the test reconstruction MSEs as a function of the number of training (80%) + validation (20%) data. For all methods, the error decreases as the number of data increases; MCAEs mostly produce the lowest errors except for some MNIST cases. Figure 4.7 shows the Peak Signal-to-Noise Ratios (PSNRs) computed with the clean test set data (the higher the better) as a function of the standard deviation of the Gaussian noise added to the training data (the number of training data is 8000). The PSNR decreases as the noise level increases; MCAEs mostly produce the highest PSNRs. Figure 4.6 shows some de-noising examples with corrupted input data of MNIST and FMNIST.

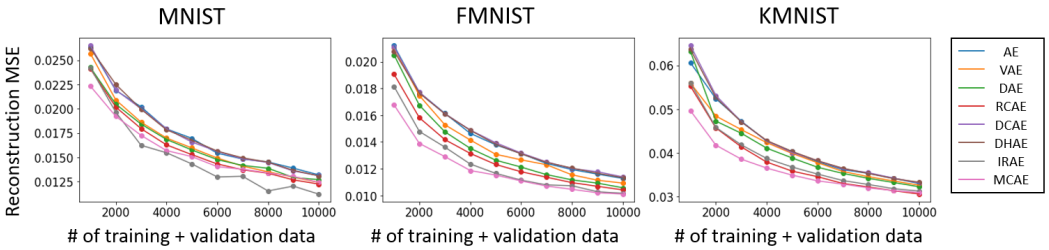


Figure 4.5: Test set MSEs as a function of the number of training (80%) + validation (20%) data, the lower the better.

SVHN & CIFAR10 Image: We compare the manifold learning performances of MCAE with other regularized autoencoders for the SVHN and CIFAR10 image datasets for both clean and corrupted training datasets. We use the convolutional and transposed convolutional neural networks for encoder and decoder with ReLU activation functions and the latent space dimensions are 64; the number of training data is 8000. For the corrupted training dataset cases, we add three different types of noise: (i) Gaussian, (ii) Shot, and (iii) Impulse noises adopted from [49]; see Figure 4.8.

Table 4.2 shows the test set MSEs for experiments with the clean training datasets, and Table 4.3 shows the PSNRs for experiments with the corrupted training datasets, where in both cases the metrics are computed with the clean test data. From the results, we note that (i) MCAE shows the second or third best results, (ii) MCAE does

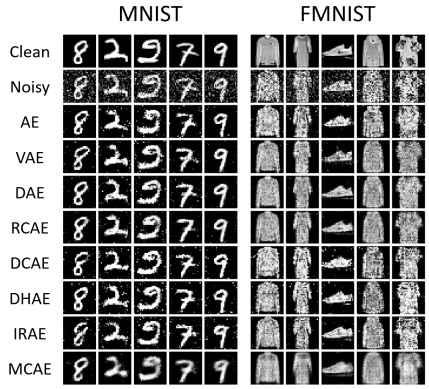


Figure 4.6: De-noising examples (noise level 0.3).

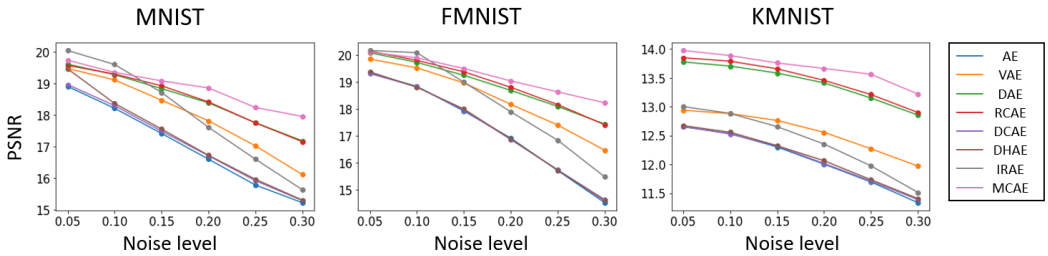


Figure 4.7: Test set Peak Signal-to-Noise Ratios (PSNR) as a function of the noise level, the higher the better.

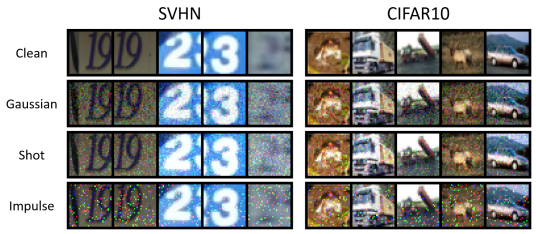


Figure 4.8: Corrupted SVHN and CIFAR10 images.

not improve the vanilla AE for the SVHN clean training dataset case, and (iii) for the corrupted training dataset cases, RCAE produces better results than the MCAE unlike the grayscale image data. Overall, compared to the grayscale image data, the

minimum curvature regularization is less effective for SVHN and CIFAR10. One possible interpretation is related to the limitation of MCAE (discussed in the conclusion section), that the SVHN and CIFAR10 manifolds have locally very different curvatures and thus it is difficult to find a proper constant regularization coefficient α in (4.3.6), because if we use a big enough α to correctly learn low curvature areas of the manifold, then high curvature areas can be overly flattened, and vice versa. Figure 4.9 shows the density plots of the log normalized local curvature of the learned manifolds by vanilla autoencoders, i.e., $\log(\kappa_i) - \overline{\log(\kappa)}, i = 1, \ldots, N$ where κ_i is the local curvature at i-th training data points and $\overline{\log(\kappa)} = 1/N \sum_i \log(\kappa_i)$, which is invariant to the scale of the mean curvature. As shown in Figure 4.9, the variance of the SVHN manifold's local curvature is bigger than those of the others, which supports the above interpretation.

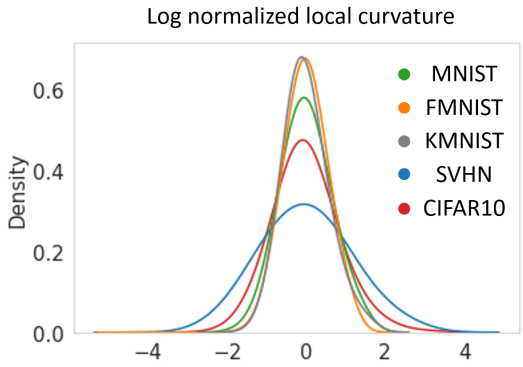


Figure 4.9: Density plots of the log-normalized local curvatures of manifolds learned by vanilla AEs.

Table 4.2: Test set MSEs of autoencoders trained with clean datasets, the lower, the better. The best and second best results are marked in red and blue, respectively.

Dataset	AE	VAE	DAE	RCAE	DCAE	DHAE	IRAE	MCAE
SVHN	0.00228	0.00461	0.00228	0.00252	0.00255	0.00233	0.00213	0.00229
CIFAR10	0.01204	0.01533	0.01204	0.01119	0.01303	0.01244	0.01176	0.01125

Table 4.3: Test data set PSNRs with various noise types, the higher, the better. The best and second best results are marked in red and blue, respectively.

Dataset	Noise type	AE	VAE	DAE	RCAE	DCAE	DHAE	IRAE	MCAE
SVHN	Gaussian	20.13	22.60	22.04	25.39	20.92	19.87	20.74	24.37
	Shot	21.11	22.49	22.97	26.20	22.41	21.23	25.18	24.73
	Impulse	19.33	19.06	20.28	23.31	19.18	19.25	19.71	20.18
CIFAR10	Gaussian	17.06	17.60	17.78	19.51	17.10	16.92	17.35	18.62
	Shot	17.18	17.46	17.87	19.52	17.26	17.18	18.49	18.64
	Impulse	16.71	16.03	16.93	18.49	16.58	16.59	16.62	17.04

4.4.4 Human Skeleton Pose Data

In this section, we evaluate the MCAE with the human skeleton pose data adopted from the NTU RGB+D dataset [50]. A human pose skeleton data onsists of 25 three-dimensional key points and thus is considered a 75-dimensional vector. There are 60 different action classes (e.g., drinking water, brushing teeth), and each action data consists of a sequence of skeleton poses. For each action class, we use randomly-selected 800 and 200 skeleton poses as training and validation data, and 9000 poses as test data. We use two-layer fully connected neural networks (512 nodes per layer) for both encoder and decoder with ELU activation functions, and the latent space dimension is 8.

Table 4.4 shows the averages and standard errors of the test data set reconstruction MSEs over 60 different action classes, the lower the better. MCAE mostly produces the lowest errors, especially by a significant margin for noisy training data cases. Figure 4.10 shows some example reconstruction results of noisy input skeleton data (noise level 0.05); MCAE shows the best de-noising results.

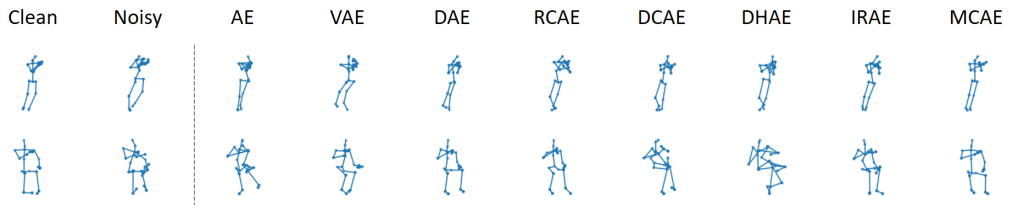


Figure 4.10: Human skeleton pose de-noising examples obtained by reconstructing noisy input data (noise level 0.05). Example poses are from the action class "eat meal".

Table 4.4: Averages and standard errors of the test data set reconstruction MSEs with various Gaussian noise of standard deviations 0.05, 0.1, the lower the better. The best and second best results are marked in red and blue, respectively. The numbers are written in units of 10^{-3}.

Noise	AE	VAE	DAE	RCAE
0	2.23 ± 0.09	2.95 ± 0.13	2.21 ± 0.09	2.17 ± 0.09
0.05	4.60 ± 0.04	4.32 ± 0.02	2.70 ± 0.01	2.98 ± 0.01
0.1	15.3 ± 0.2	13.5 ± 0.2	5.54 ± 0.17	7.50 ± 0.16
	DCAE	DHAE	IRAE	MCAE
0	2.25 ± 0.09	2.22 ± 0.09	2.08 ± 0.09	2.11 ± 0.09
0.05	3.92 ± 0.03	4.07 ± 0.03	2.93 ± 0.02	2.20 ± 0.01
0.1	11.7 ± 0.2	12.5 ± 0.2	12.7 ± 0.2	3.09 ± 0.15

4.5 Conclusion

In this thesis, we have proposed a *minimum extrinsic curvature principle* for manifold regularization and developed a **Minimum Curvature Autoencoder (MCAE)**, by formulating a coordinate-invariant (reparametrization-invariant) hence geometrically correct extrinsic curvature measure. Our experiments show that the minimum curvature regularization can improve manifold learning accuracy for both noisy and small training datasets. The degree to which the performance is improved depends on the datasets, and especially for the grayscale image and human skeleton pose datasets, the MCAE outperforms the existing methods by a significant margin.

Limitations and Future Directions: In the current implementation of MCAE, the manifold's extrinsic curvature is minimized globally by using equal weights for all points. However, for manifolds that have locally very different curvatures, it is difficult to find a proper weight parameter α in (4.3.6). Ideally, low and high curvature areas of the manifold need to be regularized with higher and lower weights, respectively. By exploiting local curvature estimation algorithms, e.g., diffusion-based method [51], developing a curvature regularization method with different local weights will be an interesting future research direction.

5

Regularized Autoencoders for Isometric Representation Learning

5.1 Introduction

Learning a good representation for high-dimensional data is one of the most fundamental problems in machine learning. A good representation should capture all essential information about the data in a parsimonious manner while filtering out all non-essential variations. Clearly what is "essential" or "non-essential" depends heavily on the end task [52], and numerous criteria have been proposed for a range of contexts, e.g., disentanglement [53, 54], clustering [26], sparsity [55], hierarchy [56], and isometric embedding [57, 58].

In this thesis we take the view that the essential information is best captured by the geometry of the data. More specifically, we adopt the manifold hypothesis as our point of departure, and further argue that a good representation should also preserve the geometry of the data manifold. That is, nearby points on the manifold should have representations in the latent representation space that are also nearby, and angles and volumes should be preserved as much as possible when moving between the data manifold and its representation space. To find such a representation, it is important to (i) learn the correct low-dimensional data manifold, and (ii) to find an optimal set of latent space coordinates that preserves the geometry of the learned data manifold.

A primary reason that autoencoders are widely used for unsupervised representation learning is that they can learn both the manifold and the latent space coordinates simultaneously during the training phase. Vanilla autoencoders trained purely to minimize reconstruction loss tend to overfit, and the learned manifolds are often inaccurate. As demonstrated in [32, 25, 26, 40, 13], by augmenting the reconstruction loss with a regularization term, manifold learning performance of autoencoders can be significantly enhanced. However, little if any consideration has been given to the

concurrent problem of learning a set of latent space coordinates that preserves the geometry of the data manifold (with the exception of FMVAE [33], which we discuss further below).

The main contribution is a regularized autoencoder that simultaneously learns the data manifold and a geometry-preserving set of latent space coordinates. For this purpose, we first formulate a hierarchy of geometry-preserving mappings. At the top of the hierarchy are isometries, which preserve distances and angles, followed by conformal maps, which preserve angles, and then area-preserving maps. Of particular relevance to this thesis are the conformal maps, which we further stratify into degree k conformal maps, with $k = 0$ corresponding to *scaled isometries*, i.e., maps that preserve angles and distances up to some scale factor.

Based on this hierarchy of mappings, we then formulate a corresponding hierarchy of regularization criteria for training the autoencoder. One of the important findings of our study is that somewhat counterintuitively, using a regularization term that measures the nearness to an isometry is in fact detrimental; such a regularization term overly constrains the mapping, resulting in a higher reconstruction loss whose effects cannot be mitigated even with adjustments to the regularization term weight. Rather, using a less stringent regularization term is more helpful. In our examples the degree zero conformal maps, or scaled isometries, seem to offer the best balance between reconstruction accuracy and model parsimony.

We note that if the exact data manifold were known in advance, then the scale factor could be pre-computed, e.g., to make the latent space and the data manifold have the same volume, in which case it would make sense to use a normalized version of the isometry measure as regularization term. When using an autoencoder for representation learning, however, the data manifold is not known a priori but rather learned together with the latent space representation. The more effective alternative, we argue, is to learn this scale together with the manifold and the latent space representation. The FMVAE of [33] is in fact the first work to adopt this approach, but the measure does not adequately capture the nearness to a scaled isometry, and is also not coordinate-invariant, limiting its performance as we show later in our experiments.

Once the data manifold and an initial set of geometry-preserving latent space coordinates are learned, it is possible to further "flatten" the latent space, by adding a postprocessing step that leads to an even more isometric set of coordinates. Specifically, we use an invertible neural network model to map the pre-trained latent space to a more isometric representation space, without incurring any further losses in reconstruction accuracy.

Experiments on diverse image and motion capture data confirm that, compared to

existing related methods, our geometrically regularized autoencoder produces more isometric representations of the data while incurring only minimal losses in reconstruction accuracy. In particular, information retrieval task experiments conducted with *CelebA* data show that data similarity measurements performed in our representation space lead to significantly improved levels of retrieval performance.

Our specific contributions can be summarized as follows:

- We define a family of coordinate-invariant regularization terms that measure how close the decoder is to being a scaled isometry;
- We propose an isometric regularization method for autoencoders that learns both the data manifold and a set of geometry-preserving latent space coordinates, all while incurring minimal losses in reconstruction accuracy;
- We propose a postprocessing flattening technique that learns a more isometric representation space without further losses in reconstruction accuracy.

5.2 A Hierarchy of Geometry-Preserving Mappings

This section introduces a hierarchy of geometry-preserving mappings between two Riemannian manifolds. Let $\mathcal{M}$ be a Riemannian manifold of dimension m with local coordinates $z \in \mathbb{R}^m$ and Riemannian metric $G(z) \in \mathbb{R}^{m\times m}$, and $\mathcal{N}$ be a Riemannian manifold of dimension n with local coordinates $x \in \mathbb{R}^n$ and Riemannian metric $H(x) \in \mathbb{R}^{n\times n}$. Let $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ be a smooth mapping, represented in local coordinates by the italic symbol $f : \mathbb{R}^m \to \mathbb{R}^n$. Its differential is denoted by the Jacobian matrix $J_f(z) := \frac{\partial f}{\partial z}(z) \in \mathbb{R}^{n\times m}$.

Intuitively, an *isometry* is a mapping between two spaces that preserves distances and angles everywhere. For a linear mapping between two vector spaces equipped with inner products, an isometry preserves the inner product everywhere. In the case of a mapping between Riemannian manifolds, $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ is an isometry if

$$G(z) = J_f(z)^T H(f(z)) J_f(z) \quad \forall z \in \mathbb{R}^m. \tag{5.2.1}$$

Sometimes, requiring a map f to be an isometry can be overly restrictive; preserving only angles may be sufficient. A *conformal map* is a mapping that preserves angles but not necessarily distances. Mathematically, $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ is conformal (or angle-preserving) if

$$G(z) = c(z) J_f(z)^T H(f(z)) J_f(z) \quad \forall z \in \mathbb{R}^m, \tag{5.2.2}$$

for some positive function $\mathrm{c} : \mathcal{M} \to \mathbb{R}$. The positive function is called the conformal factor.

Conformal maps can be further categorized by the polynomial degree of $c(z)$. A conformal map of degree zero, i.e., one in which $c(z)$ is constant, sits one level below the isometric mapping and is defined formally as any mapping f for which a positive scalar constant c satisfying

$$G(z) = cJ_f(z)^T H(f(z))J_f(z) \quad \forall z \in \mathbb{R}^m \tag{5.2.3}$$

can be found. Such a map not only preserves angles but also scaled distances; for this reason we shall also refer to a degree zero conformal map as a *scaled isometry*. Area-preserving maps also can be placed within this hierarchy, but for the purposes of this thesis our focus will be exclusively on isometric and conformal mappings.

5.3 A Coordinate-Invariant Relaxed Distortion Measure

The goal of this section is to design a coordinate-invariant functional $\mathcal{F}$ that measures the proximity of the mapping $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ to a scaled isometry. Section 5.3.1 shows how to construct coordinate-invariant functionals for a smooth mapping between two Riemannian manifolds, while Section 5.3.2 introduces a simple technique to define coordinate-invariant distortion measures that measure how close the mapping is to being an isometry. Section 5.3.3 defines a family of coordinate-invariant *relaxed* distortion measures that measure how close the mapping is to being a scaled isometry.

In order to extend the discussion of traditional distortion measures to more general cases (we will later use a probability measure), we consider a positive measure ν on $\mathcal{M}$ absolutely continuous to the Riemannian volume form,[1] the spaces of interest will then be limited to the support of ν rather than the entire manifold $\mathcal{M}$.

5.3.1 Coordinate-Invariant Functionals on Riemannian Manifolds

We begin this section by reviewing how to construct coordinate-invariant functionals for a smooth mapping $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ [46]. At a point $\mathrm{z} \in \mathcal{M}$, consider the characteristic values of the pullback metric $J_f^T H J_f$ relative to the metric G of $\mathcal{M}$, i.e., the

[1] Given a measurable space $\mathcal{M}$, a measure ν is absolutely continuous to the Riemannian volume measure Vol such that $d\mathrm{Vol}(z) = \sqrt{\det G(z)}\, dz$, if $\mathrm{Vol}(A) = 0$ implies $\nu(A) = 0$ for any measurable set $A \subset \mathcal{M}$.

m eigenvalues $\lambda_1(z), ..., \lambda_m(z)$ of $J_f(z)^T H(f(z)) J_f(z)^T G^{-1}(z)$. Apart from their order, these eigenvalues are intrinsically associated with $J_f^T H J_f$ and G, i.e., they are invariant under coordinate transformations.[2]

Let $S(\lambda_1, ..., \lambda_m)$ be any symmetric function (i.e., a function whose value is invariant with respect to permutations of its arguments) of the m eigenvalues. Then the integral

$$\mathcal{I}_S(f) := \int_{\mathcal{M}} S(\lambda_1(z), ..., \lambda_m(z))\, d\nu(z) \tag{5.3.4}$$

is an intrinsic quantity, i.e., coordinate-invariant.

In this thesis we introduce a new family of coordinate-invariant functionals that can be used to formulate measures for scaled isometries in a more natural way. Consider a symmetric function S such that the above integral $\mathcal{I}_S(f) \neq 0$, and let $S'(\lambda_1, ..., \lambda_m)$ be any symmetric function. The following integral

$$\int_{\mathcal{M}} S'(\frac{\lambda_1(z)}{\mathcal{I}_S(f)}, ..., \frac{\lambda_m(z)}{\mathcal{I}_S(f)})\, d\nu(z) \tag{5.3.5}$$

is then an intrinsic quantity, since the eigenvalues and $\mathcal{I}_S(f)$ are both intrinsic quantities.

5.3.2 Distortion Measures of Isometry

Recall that $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ is a local isometry at z if $\lambda_i(z) = 1$ for $\forall i$, and a global isometry if f is a local isometry everywhere [46]. In this section we introduce a simple technique to define a family of distortion measures that measure the proximity to an isometry. Consider any convex function $h : \mathbb{R} \to [0, \infty)$ such that $h(1) = 0$, $h'(\lambda) = 0$ iff $\lambda = 1$, and define a symmetric function $S(\lambda_1, ..., \lambda_m) = \sum_{i=1}^m h(\lambda_i)$. Then the coordinate-invariant functional

$$\int_{\mathcal{M}} \sum_{i=1}^{m} h(\lambda_i(z))\, d\nu(z) \tag{5.3.6}$$

is a global measure of distortion (restricted to the support of ν). Since the integrand in the above functional locally measures the deviation of the mapping f from an isometry, its integral also serves as a global measure of distortion. Popular choices include $h(\lambda) = (1 - \lambda)^2$ and $h(\lambda) = (\log(\lambda))^2$.

[2] Eigenvalues of $J_f^T H J_f G^{-1}$ remain the same under coordinate transformations $z \mapsto \phi(z), x \mapsto \psi(x)$.

5.3.3 A Relaxed Distortion Measure and Scaled Isometry

The goal of this section is to design a family of coordinate-invariant functionals $\mathcal{F}$ that measure how far the mapping $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ is from being a scaled isometry (over the support of ν). We refer to these as *relaxed distortion measures*, in the sense that a larger set of mappings are minimizers of the relaxed distortion measure than the original distortion measure.

Given mappings $\mathrm{f}, \mathrm{f}' : \mathcal{M} \to \mathcal{N}$ (denoted f and f' in local coordinates) with respective Jacobians J_f, $J_{f'}$, the desired properties of $\mathcal{F}$ are as follows:

(i) $\mathcal{F}(f) \geq 0$;

(ii) $\mathcal{F}(f) = 0$ if and only if $\lambda_i(z) = c$ for $\forall i$, $\forall z \in \mathrm{Supp}(\nu)$, and for some $c > 0$;

(iii) $\mathcal{F}(f) = \mathcal{F}(f')$ if $J_f^T H J_f = c J_{f'}^T H J_{f'}$ for $\forall z \in \mathrm{Supp}(\nu)$, and for some $c > 0$,

where $\lambda_i(z)$ denotes the eigenvalues of $J_f^T(z) H(f(z)) J_f(z) G^{-1}(z)$ and $\mathrm{Supp}(\nu)$ is the support of ν. The first condition is to ensure a minimum of zero, while the second condition is to make any scaled isometry (restricted to the support of ν) be a minimizer. Although the first and second conditions are sufficient to use $\mathcal{F}(f)$ as a measure of the proximity of f to a scaled isometry, we impose an additional third condition to make the measure more natural in the following sense: since the measure should not a priori favor a particular scale for the pullback metric, if the pullback metrics are equivalent up to some scale, then these should be treated equivalently.

Our core idea for constructing such a measure is to use the newly proposed family of coordinate-invariant functionals (5.3.5) with the technique used in (5.3.6) as follows:

$$\mathcal{F}(f) := \int_{\mathcal{M}} \sum_{i=1}^{m} h\left(\frac{\lambda_i(z)}{\int_{\mathcal{M}} S(\lambda_1(z), ..., \lambda_m(z))\, d\nu(z)}\right) d\nu(z) \tag{5.3.7}$$

where h is some convex function and S is some symmetric function. This functional automatically satisfies the first condition. To satisfy the second and third conditions, the symmetric function S must satisfy some further conditions:

Proposition 5.3.1. The functional in (5.3.7) satisfies the second and third conditions if ν is a finite measure, $S(k\lambda_1, ..., k\lambda_m) = kS(\lambda_1, ..., \lambda_m)$, and $S(1, ..., 1) = 1/\nu(\mathcal{M})$.
Proof: See Appendix C.

5.4 Isometric Representation Learning

We now introduce a regularized autoencoder with a practical form of the relaxed coordinate-invariant distortion measure. We also describe the additional postprocessing step for further flattening the latent space. Finally, we introduce some relevant implementation details.

5.4.1 Isometric Regularization with the Relaxed Distortion Measure

Denote a parametric encoder function by $g_\phi : \mathbb{R}^D \to \mathbb{R}^m$ and decoder function by $f_\theta : \mathbb{R}^m \to \mathbb{R}^D$. We consider the latent space $\mathbb{R}^m$ with coordinates z assigned with the identity metric $G(z) = I$ and the data space $\mathbb{R}^D$ with coordinates x assigned with the metric $H(x)$. We denote the data distribution in $\mathbb{R}^D$ by $x \sim P_D$ and the distribution of the encoded data in $\mathbb{R}^m$ by $z \sim P_\phi$ where $z = g_\phi(x)$.

First, to construct a relaxed distortion measure of f_θ, we need to select a positive measure ν. Considering ν as a probability measure P_ϕ and replacing the integrals $\int_{\mathcal{M}} d\nu(z)$ in (5.3.7) by expectations $\mathbb{E}_{z\sim P_\phi}$, we get the following expression:

$$\mathcal{F}(f_\theta; P_\phi) := \mathbb{E}_{z\sim P_\phi}[\sum_{i=1}^{m} h(\frac{\lambda_i(z)}{\mathbb{E}_{z\sim P_\phi}[S(\lambda_1(z), ..., \lambda_m(z))]})], \tag{5.4.8}$$

where $\lambda_i(z)$ are the eigenvalues of the pullback metric $J_{f_\theta}^T(z)H(f_\theta(z))J_{f_\theta}(z)$.

Then, we propose a regularized autoencoder where the loss function consists of the following two terms i) loss function $\mathcal{L}(\theta, \phi)$ for manifold learning (e.g., reconstruction error) and ii) regularization term $\mathcal{F}(f_\theta; P_\phi)$ for learning a scaled isometric decoder:

$$\min_{\theta,\phi} \mathcal{L}(\theta, \phi) + \alpha\mathcal{F}(f_\theta; P_\phi). \tag{5.4.9}$$

Training an autoencoder to minimize (5.4.9) is referred to as **Isometric Regularization (IR)**.

In practice, the computation of and back-propagation through $\mathcal{F}(f_\theta; P_\phi)$ that includes the computation of the entire Jacobian of f_θ requires sufficient memory and computational cost. Instead, with an appropriate choice of convex function h and symmetric function S, the measure $\mathcal{F}(f_\theta; P_\phi)$ can be efficiently estimated by using the more easily computed Jacobian-vector and vector-Jacobian products and Hutchinson's stochastic trace estimator [48]:

Proposition 5.4.1. Let $h(\lambda) = (1-\lambda)^2$ and $S(\lambda_1, ..., \lambda_m) = \sum_{i=1}^{m} \lambda_i/m$, then

$$\mathcal{F}(f_\theta; P_\phi) = \mathbb{E}_{z\sim P_\phi}[\sum_{i=1}^{m}(\frac{\lambda_i(z)}{\mathbb{E}_{z\sim P_\phi}[\sum_i \lambda_i(z)/m]} - 1)^2] = m^2 \frac{\mathbb{E}_{z\sim P_\phi}[\mathrm{Tr}(H_\theta^2(z))]}{\mathbb{E}_{z\sim P_\phi}[\mathrm{Tr}(H_\theta(z))]^2} - m, \tag{5.4.10}$$

where $\lambda_i(z)$ means the eigenvalues of $H_\theta(z) := J_{f_\theta}^T(z)H(f_\theta(z)))J_{f_\theta}(z)$. Proof: See Appendix C.

5.4.2 Latent Space Flattening

In the isometric regularization approaches, autoencoders are trained to minimize both manifold learning loss and geometric regularization term; hence there exists an inherent tradeoff. We introduce a postprocessing step to further flatten the pre-trained latent space to a more isometric representation space without incurring any further losses in reconstruction accuracy.

Given a trained encoder $g_\phi : \mathbb{R}^D \to \mathbb{R}^m$ and trained decoder $f_\theta : \mathbb{R}^m \to \mathbb{R}^D$, consider an invertible map $i : \mathbb{R}^m \to \mathbb{R}^m$ such that $z' = i(z)$ and the composition $f_\theta \circ i^{-1} : \mathbb{R}^m \to \mathbb{R}^D$. The map i transforms the pre-trained latent space to a new set of latent coordinates without affecting the reconstruction accuracy. This gives an additional degree of freedom to find a more isometric representation space. Based on this idea, we formulate the latent space flattening problem as follows:

$$\min_i \mathcal{F}(f_\theta \circ i^{-1}; P_i') + \beta\, \mathbb{E}_{z\sim P_\phi}[\|i(z)\|^2], \tag{5.4.11}$$

where $z' \sim P_i'$ is the distribution of the encoded data in $\mathbb{R}^m$ ($z' = i \circ g_\phi(x)$) and the second term is the regularization term with coefficient β added to prevent $i(z)$ from diverging. In this thesis, we use the invertible deep neural network *RealNVP* [59] for i.

5.4.3 Implementation Details

We now explain some relevant implementation details of our approach. We use a Gaussian encoder $q_\phi(z|x)$ and decoder $p_\theta(x|z)$, and treat the mean of $p_\theta(x|z)$ as $f_\theta(z)$. We use the negative evidence lower bound for $\mathcal{L}(\theta, \phi)$ from the VAE with the unit Gaussian prior distribution [25]. We assume the identity metric for the data space, i.e., $H(x) = I$. We use the particular combination of h and S described in Proposition 5.4.1. In cases when the latent space dimension is sufficiently small, it becomes feasible to compute the full metric and use other combinations. As an example, when the dimension is two, we use another popular choice $h(\lambda) = (\log(\lambda))^2$.

Augmented Distribution: The relaxed distortion measure is defined as the expectation over the encoded data distribution P. However, the influence of the measure is then limited to regions where data is available; in practice, data augmentation can resolve this issue. Following the FMVAE [33], we use the modified mix-up data-augmentation method with $\eta > 0$, where P is augmented by $z = \delta z_1 + (1-\delta)z_2$ such that $z_i \sim P, i = 1, 2$, where δ is uniformly sampled from $[-\eta, 1+\eta]$. Sampling from the augmented latent space data distribution is denoted as $z \sim P_Z$.

The pseudocode is available in Appendix C.

5.5 Experiments

Throughout this section we use isometric regularization on VAE [25, 60] which we refer to as the *Isometrically Regularized VAE (IRVAE)*. We then train the *Flattening Module (FM)* in a post hoc manner, which we denote by *IRVAE + FM*. In Section 5.5.1, we show that (i) IRVAE can effectively learn the isometric representation with a minimal loss in reconstruction accuracy compared to vanilla VAE and FMVAE [33], and (ii) IRVAE + FM leads to a more isometric representation without any loss in reconstruction accuracy. In Section 5.5.2, through an unsupervised human face retrieval task, we show that measuring data similarity in our isometric representation spaces significantly improves retrieval performance. A mathematical comparison of IRVAE with FMVAE and experimental details are given in Appendix C.

5.5.1 Isometric Representation with Minimal Loss in Reconstruction Accuracy

We first introduce some evaluation metrics that measure (i) the accuracy of the learned manifolds, and (ii) how isometric the latent space is (i.e., how close the pullback metric $M(z) := J_f^T(z)J_f(z)$ is to $\{cI | c \in (0, \infty)\}$). These metrics are computed over the test datasets.

The accuracy of the learned manifolds is measured in terms of the mean square reconstruction errors. To evaluate $M(z)$, we introduce two different metrics that are scale-invariant (i.e., $M(z)$ and $kM(z)$ have the same values). First, the *Variance of the Riemannian metric (VoR)* is defined as the mean square distance from $M(z)$ to $\bar{M} := \mathbb{E}_{z\sim P_Z}[M(z)]$, where we use the affine-invariant Riemannian distance d, i.e., $d^2(A, B) = \sum_{i=1}^{m} (\log \lambda_i (B^{-1}A))^2$ [61, 62]. Second, we use the *Mean Condition Number (MCN)* of the Riemannian metric, where the condition number of $M(z)$ is the ratio between the maximum eigenvalue and minimum eigenvalue. Both metrics are computed by sampling a sufficiently large number of points from P_Z.

VoR measures how the Riemannian metric is spread out from $\bar{M}$, which becomes zero when the metric is constant for all z in the support of P_Z. While VoR measures the spatial variance of the metric in z, MCN measures how much the metric is isotropic. MCN becomes one when $M(z) \propto I$ for all z in the support of P_Z.

For IRVAE and FMVAE, there are natural tradeoffs between MSE versus VoR and MCN. If we use higher weights α for the regularization terms, the representation becomes more isometric with losses in reconstruction accuracy. In order to compare the algorithms in a regularization coefficient α-invariant manner, we compare the tradeoff curves, i.e., MSE as a function of VoR and MCN obtained by using varying regularization coefficients α.

Sections 5.5.1.1 and 5.5.1.2 use the *MNIST* image and *CMU motion capture* data, respectively, with two-dimensional latent spaces. Additional experimental results including experiments on more diverse image data (*MNIST, FMNIST, SVHN, CIFAR-10*) are provided in Appendix C.

5.5.1.1 MNIST

Figure 5.1 shows tradeoff curves of FMVAE and IRVAE trained on MNIST data with two-dimensional latent spaces, some example reconstructed images, and latent space representations with equidistance plots. We note that the tradeoff curves of IRVAE (orange) are below those of FMVAE (blue), meaning that the IRVAE learns more isometric representations at any level of reconstruction accuracy. Cases A and B produce good reconstruction results, but non-homogeneous and non-isotropic equidistance plots since lower regularization coefficients are used. For cases F, I, opposite results are obtained.

Figure 5.2 shows tradeoff curves and latent space representations for IRVAE + FM (under the same experimental setting as above). IRVAE + FM results in more isometric representations than IRVAE, with no losses in MSE. In particular, for cases A and B, which have the lowest MSE but the highest VoR and MCN, the flattener significantly lowers both VoR and MCN. These improvements can be qualitatively seen by comparing the equidistance plots of A and B in Figure 5.1 and Figure 5.2.

5.5.1.2 CMU Motion Capture Data

We use a subset of the CMU motion capture data by selecting four classes: walking, jogging, balancing, and punching, with two-dimensional latent spaces. The motion data in

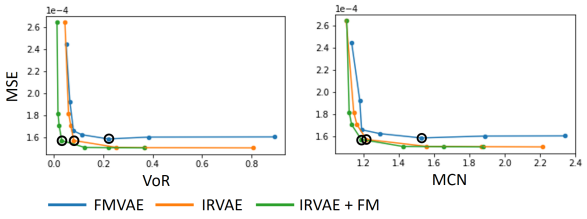


Figure 5.3: The tradeoff curves for FMVAE, IRVAE, IRVAE + FM trained with the pose data.

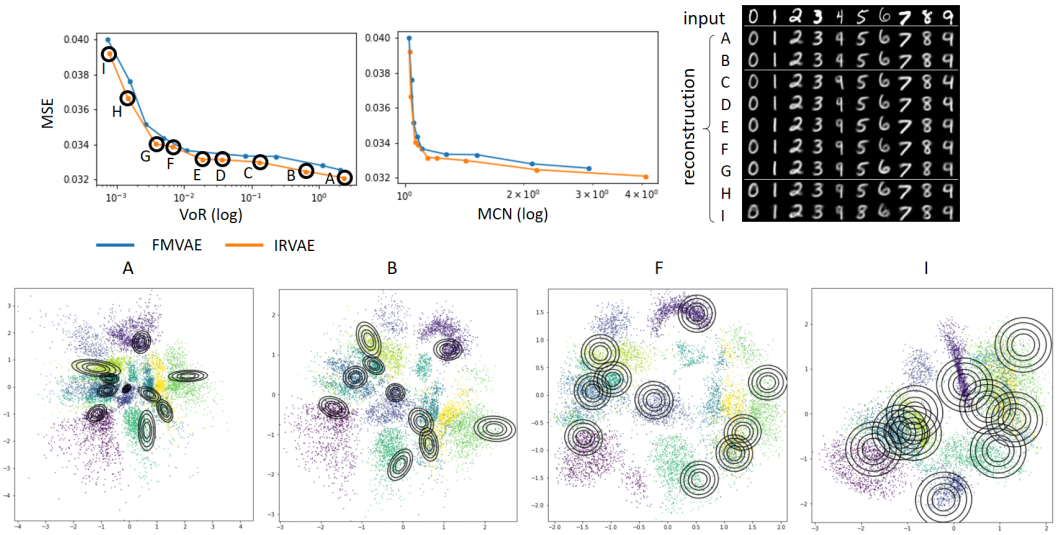


Figure 5.1: *Top:* The MSE and VoR, MCN tradeoff curves, and some example reconstructed images produced by IRVAE trained with various regularization coefficients. *Bottom:* Two-dimensional latent space representations with some equidistance plots whose centers consist of a randomly selected data point z_c from each class for A, B, F, I.The equidistance plots are $\{z|(z-z_c)^T J_f^T(z_c)J_f(z_c)(z-z_c) = k$ for $k > 0$. (The more homogeneous and isotropic, the better.)

each class is a sequence of pose data, where each pose data is represented by a 50-dimensional vector of joint angles. We use a total of 10,000, 2,000 and 2,000 pose data items for training, validation, and test, respectively.

Figure 5.3 shows the tradeoff curves for FMVAE, IRVAE, and IRVAE + FM. The results of VAE are omitted from the figure since the MSE, VoR, and MCN are too large to be plotted together with the other models (MSE: 1.72×10^{-4}, VoR: 4.86, MCN: 20.3). The tradeoff curves for IRVAE are far below compared to those for FMVAE, while the IRVAE + FM results in a more isometric representation without any loss in MSE.

Figure 5.4 shows the latent space representations with equidistance ellipses. For a fair comparison, we select models with similar reconstruction accuracy (the selected

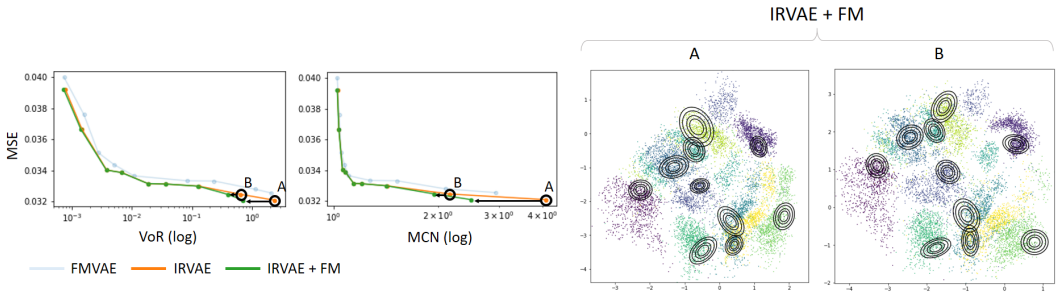


Figure 5.2: Tradeoff curves for FMVAE, IRVAE, and IRVAE + FM, and two-dimensional latent space representations with some equidistance plots (under the same experimental setting as Figure 5.1).

regularization coefficients are marked by the black circles in Figure 5.3). We observe that the ellipses for IRVAE and IRVAE + FM are much more homogeneous (blue) than those for VAE and FMVAE.

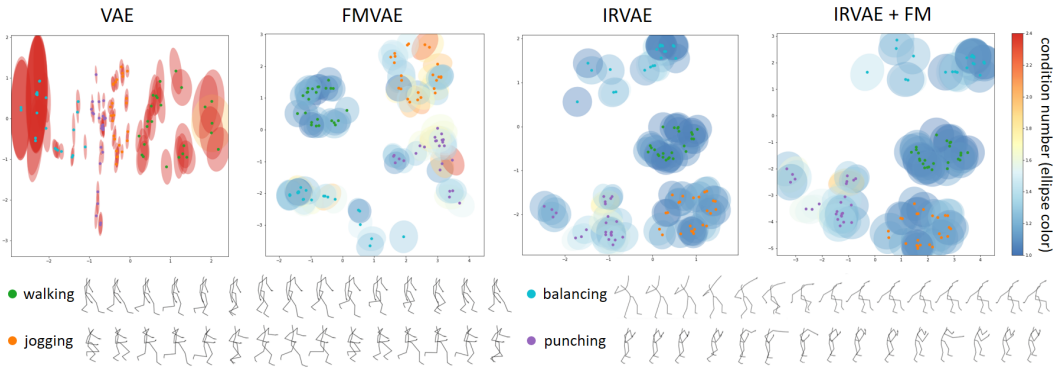


Figure 5.4: Latent spaces and equidistance ellipses for VAE, FMVAE, IRVAE, and IRVAE + FM (the redder the ellipse, the larger the condition number).

5.5.2 Unsupervised Human Face Retrieval

We also demonstrate the effectiveness of our approach to image-to-image retrieval of human faces. We consider the retrieval of face images that have a queried set of attributes, in which the query is also given as a set of face images that share some visual attributes. For example, if a user provides a set of photos that contain smiling faces but differ in other attributes, a retrieval algorithm is expected to return a set of

smiling face images from the database. In this setting, a user communicates with a retrieval system by providing examples, avoiding the dependence on human language.

We focus in particular on an unsupervised scenario in which a retrieval algorithm is built without any annotation. Unsupervised representation learning methods are trained on CelebA [63], which contains 182,637 training images and 19,962 test images. Each image in CelebA has 64×64×3 pixels and contains an aligned human face with binary annotations on 40 attributes. The attribute annotations are only used to evaluate retrieval performance.

We refer to attributes that query images have in common as query attributes. We experiment with different settings in which there is a single query attribute (single attribute retrieval; SAR) and also with two query attributes (double attribute retrieval; DAR). We use all images that have given query attributes in the training set as query images, and retrieve the top K similar images from the test set based on the summed cosine similarity to the query images.

The quality of a retrieval is measured using precision at K for $\mathrm{K} = 1, 5, 10, 20$ (P@K). P@K denotes the ratio of images that have query attributes among the K retrieved images. For SAR, P@K are averaged over the 40 attributes. For DAR, there are 780 possible combinations of two attributes. We select a combination of attributes if the images that have those attributes account for more than one percent and less than fifty percent of the total test data; thus P@K are averaged over 487 selected combinations.

Retrieval is performed in the latent space of the representation learning algorithms. Algorithms to be compared include (i) unsupervised representation learning methods VAE [25], FMVAE [33], IRVAE (ours), and IRVAE + FM (ours), whose latent space dimensions are 128, (ii) a neural network (ResNet-50) pre-trained on ImageNet [64], and (iii) a supervised learning method (binary relevance) [65]. The performance of the pre-trained network and the supervised learning method serve as lower and upper bounds for the unsupervised methods, respectively.

Table 5.1 lists the retrieval performance for the various cases. IRVAE outperforms FMVAE, which in turn outperforms VAE. FM improves the performance of IRVAE for most cases with only one exception, P@1 score in SAR. More detailed results on SAR are included in Appendix C. Some example image retrieval results are visualized in Figure 5.5.

Table 5.1: Precision at K retrieved images, i.e., P@K, averaged over the attributes. The best results among unsupervised methods are colored red, while the second best results are colored blue.

Method	**P@1**	**P@5**	**P@10**	**P@20**	**P@1**	**P@5**	**P@10**	**P@20**
	Single Attribute Retrieval				*Double Attribute Retrieval*			
VAE	60.0	60.5	61.0	58.4	39.8	38.7	37.0	34.7
FMVAE	67.5	66.0	64.0	62.6	41.5	39.1	38.8	36.8
IRVAE (ours)	**82.5**	**72.0**	**71.5**	**69.0**	**56.1**	**50.8**	**47.2**	**44.2**
IRVAE + FM (ours)	**75.0**	**79.0**	**75.3**	**74.1**	**57.3**	**54.0**	**51.4**	**49.4**
Pre-trained (ResNet-50)	35.0	28.5	23.5	23.9	18.3	11.0	8.7	9.2
Supervised (BR)	87.5	82.5	80.3	80.8	66.9	65.2	63.7	62.6

5.6 Conclusion

We have formulated the problem of learning the manifold simultaneously with a set of optimal latent space coordinates that preserve the geometry of the learned manifold. We have introduced a hierarchy of geometry-preserving mappings between two Riemannian manifolds (e.g., isometry, conformal mapping of degree k, area-preserving mappings) and defined a family of coordinate-invariant relaxed distortion measures that measure the proximity of the mapping to a scaled isometry (i.e., conformal mapping of degree 0). Finally, two algorithms, isometric regularization and latent space flattening, have been proposed. We have verified the efficacy of our methods with diverse image and motion capture data, and through a human face retrieval task with CelebA data.

We believe the algorithm can be further enhanced in a number of different ways. The current implementation of IRVAE and IRVAE + FM assumes the identity metric in the ambient data space, i.e., $H(x) = I$. Although this is a reasonable choice in a fully unsupervised setting, we think that domain-specific knowledge or a few labels can be leveraged to define better $H(x)$. In addition, instead of the mix-up data augmentation method used in this thesis, developing a principled approach for defining P_Z will be an interesting future research direction.

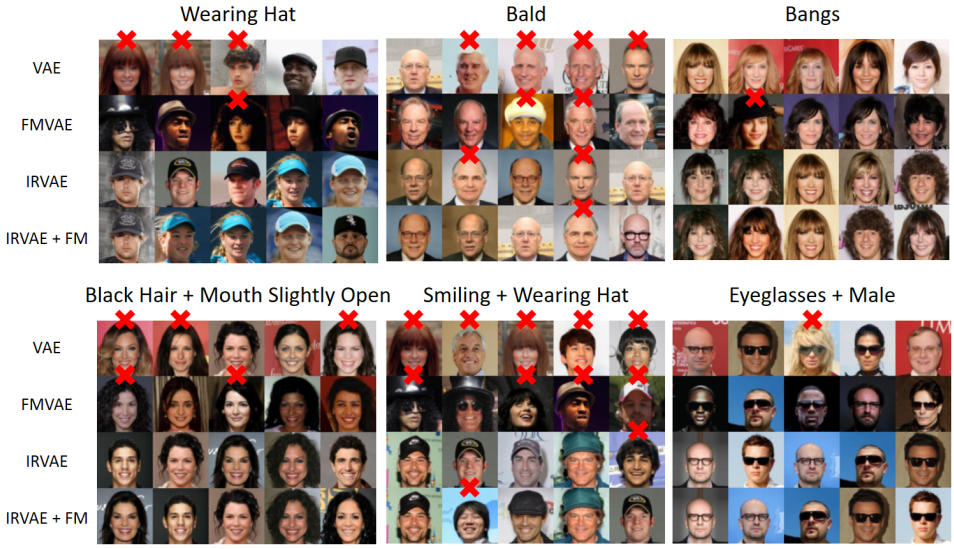


Figure 5.5: Some example image retrieval results (top 5 images). Common attributes of query image sets are written above the figures. Higher rank images are located left.

6

A Statistical Manifold Framework for Point Cloud Data

6.1 Introduction

Many machine learning problems involve data sets in which each data point is a point cloud in $\mathbb{R}^D$. For example, to measure the similarity between two shapes, point cloud representations of the two shapes can be obtained with a depth camera, and a distance metric on the space of point clouds, e.g., the Hausdorff distance, the chamfer distance [66], or earth mover distance [67] used to measure their similarity.

The distance metric measures just one aspect of point cloud data; other applications may require more advanced concepts and tools. For example, in the case of a moving point cloud, one may seek any number of quantities such as the velocity, acceleration, relative heading direction, or the swept volume. For multiple point cloud samples, one may seek a measure of their dispersion, e.g., the covariance or higher moments.

In fact, a growing number of applications involving point cloud data require a means of measuring not only distances, but also angles, volumes, derivatives, and other advanced geometric and analytical concepts. In principle one could choose some arbitrary coordinates to parameterize the space of point clouds, and extend the usual Euclidean notions of angles, volumes, and derivatives, but such an approach would not only be ad hoc, but likely not be invariant with respect to the choice of coordinates.

To formulate and quantify these concepts in a coordinate-invariant, geometrically meaningful way, as a first contribution we develop a *Riemannian geometric* framework for point cloud data. The key idea behind our approach draws upon information geometry [68, 69]: by interpreting each point in a point cloud as a sample drawn from some known probability density, the space of point cloud data can be given the structure of

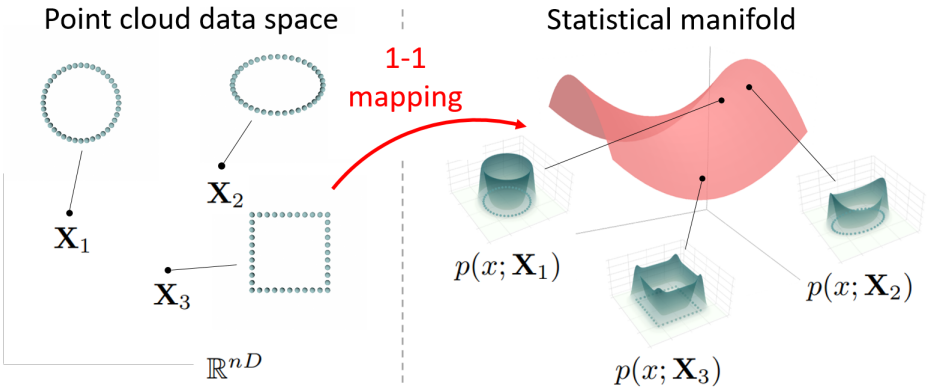


Figure 6.1: Illustration of statistical manifold obtained from the 1-1 mapping between the space of point cloud data and the space of probability density functions.

a *statistical manifold* – each point on this manifold represents a point cloud – with the Fisher information metric acting as a natural Riemannian metric. Under some mild assumptions, i.e., the number of points in a point cloud is fixed, and all points are distinct, a one-to-one mapping between the space of point clouds and probability densities can be constructed. That is, a point cloud $\mathbf{X} = \{x_1, ..., x_N | x_i \in \mathbb{R}^D\}$ is mapped to a density function $p(x;\mathbf{X})$ on $\mathbb{R}^D$ in a 1-1 fashion as illustrated in Figure 6.1.

We remark that the idea of interpreting each point in a point cloud as a sample drawn from some given probability density is well-known, and has been applied to problems ranging from point set registration [70, 71, 72, 73, 74, 75, 76] to point cloud de-noising [77, 78]. These applications, however, only require a similarity measure between point clouds, typically formulated in terms of some divergence measure.

Two autoencoder applications of our framework are presented: (i) smoothly deforming one 3D object (a cylinder) into another (a cone), and (ii) learning an optimal set of latent space coordinates for point cloud data that best preserves distances and angles. In the former case, a pre-trained autoencoder is used to encode two 3D point clouds – one representing the cylinder, the other the cone – and the minimal geodesic with respect to the natural Riemannian metric is then constructed between these two objects. The shape evolution obtained for this Riemannian metric is seen to be far more natural and intuitive than that obtained for the straight line interpolant in latent space.

In the second application, we use the statistical manifold framework to find a set of distortion minimizing latent space coordinates, in the sense that Euclidean straight

lines in the latent space closely approximate minimal geodesics on the statistical manifold. Such a set of coordinates offers a more discriminative representation for the data manifold [33, 79] that results in, e.g., higher linear SVM classification accuracy vis-á-vis existing state-of-the art methods. Experiments are carried out with both synthetic and standard benchmark datasets (*ShapeNet* [80], *ModelNet* [81]).

6.2 Statistical Manifold Framework for Point Cloud Data

We begin this section with some information geometric preliminaries. A *statistical manifold* is an infinite-dimensional Riemannian manifold each of whose points is a probability density, with the Fisher information metric acting as a natural Riemannian metric. Finite-dimensional statistical manifolds can be obtained by considering a family of parametric probability density functions:

Definition 6.2.1. Given an m-dimensional differentiable manifold Θ and a smooth 1-1 map from Θ to the space of probability density functions $\boldsymbol{\theta} \mapsto p(x;\boldsymbol{\theta})$, the image of this mapping, denoted $\mathcal{S} := \{p(x;\boldsymbol{\theta})|\boldsymbol{\theta} \in \Theta\}$, is an m-dimensional statistical manifold.

Let $\theta = (\theta^1, ..., \theta^m) \in \mathbb{R}^m$ be local coordinates for $\boldsymbol{\theta} \in \Theta$, which by trivial extension also act as local coordinates for $\mathcal{S}$. Throughout we use italics to represent local coordinates, e.g., $\boldsymbol{\theta} \in \Theta$ has local coordinates $\theta \in \mathbb{R}^m$. In this coordinate system, elements g_{ij} of the Fisher information metric $G(\theta) \in \mathbb{R}^{m\times m}$ are given by

$$g_{ij}(\theta) := \int p(x;\theta) \frac{\partial \log p(x;\theta)}{\partial \theta^i} \frac{\partial \log p(x;\theta)}{\partial \theta^j}\, dx, \tag{6.2.1}$$

where $i, j = 1, \ldots, m$. The length of a curve on $\mathcal{S}$ parametrized by $\theta(t)$, $t \in [0, T]$, can then be computed as the integral $\int_0^T ds$, where the infinitesimal length ds on $\mathcal{S}$ is given by $ds^2 = d\theta^T G(\theta) d\theta$. Further details on statistical manifolds and the Fisher information metric can be found in, e.g., [68, 82, 83, 16].

With the above statistical manifold preliminaries, we now construct a Riemannian geometric structure for the space of point cloud data. Section 6.2.1 defines a statistical manifold from the point cloud data, while Section 6.2.2 uses the Fisher information metric to construct a Riemannian metric for point cloud data. To keep the definitions and ensuing results simple, we assume throughout that all point clouds consist of exactly n distinct points in $\mathbb{R}^D$, i.e., each point cloud is of the form

$$\mathbf{X} = \{x_1, ..., x_n \mid x_i \in \mathbb{R}^D, x_i \neq x_j \text{ if } i \neq j\}. \tag{6.2.2}$$

The set of all point clouds is denoted $\mathcal{X}$. Later we discuss methods for dealing with

point clouds that do not satisfy these assumptions. Proofs of all propositions in this section can be found in Appendix D.

6.2.1 Statistical Manifold of Point Cloud Data

Given a point cloud $\mathbf{X}$, a parametric probability density function $p(x;\mathbf{X})$ can be defined in terms of a positive kernel function $\mathbf{X}$ itself as the parameter [84, 85]:

Definition 6.2.2. Given a positive kernel function $K : \mathbb{R}^D \to \mathbb{R}$ that satisfies the following constraint $\int_{\mathbb{R}^D} K(u)\,du = 1$, and a $D \times D$ symmetric positive-definite matrix Σ (the bandwidth matrix), the kernel density estimate

$$p(x;\mathbf{X}) := \frac{1}{n\sqrt{|\Sigma|}} \sum_{i=1}^{n} K(\Sigma^{-\frac{1}{2}}(x - x_i)) \tag{6.2.3}$$

is said to be a statistical representation of the point cloud $\mathbf{X} \in \mathcal{X}$. The set of statistical representations is denoted $\mathcal{S} := \{p(x;\mathbf{X}) \mid \mathbf{X} \in \mathcal{X}\}$.

To ensure that $\mathcal{S}$ is a statistical manifold, recall from Definition 6.2.1 that the following two conditions need to be satisfied: (i) $\mathcal{X}$ is a differentiable manifold; (ii) A 1-1 mapping $h : \mathcal{X} \to \mathcal{S}$, $h(\mathbf{X}) = p(x;\mathbf{X})$ must be defined. The first condition can be satisfied with the "distinct points" assumption:

Proposition 6.2.1 (Corollary 2.2.11. in [86])**.** The set of point clouds in which each point cloud is a set of n distinct points of dimension D, is an nD-dimensional differentiable manifold.[1]

To satisfy the second condition, additional assumptions are needed. The following proposition provides a sufficient condition for h to be 1-1:

Proposition 6.2.2. If the kernel function

$$\Psi(x, y) = K(\Sigma^{-\frac{1}{2}}(x - y)) \tag{6.2.4}$$

is strictly positive-definite,[2] then the mapping $h : \mathcal{X} \to \mathcal{S}$ given by

$$h(\mathbf{X})(x) = \frac{1}{n\sqrt{|\Sigma|}} \sum_{i=1}^{n} K(\Sigma^{-\frac{1}{2}}(x - x_i)) \tag{6.2.5}$$

[1] Without the distinct points assumption, the set of point clouds is no longer a manifold, but only an orbifold that is locally a finite group quotient of a Euclidean space.

[2] A kernel function $\Psi(\cdot,\cdot)$ is said to be *strictly positive-definite* if the matrix $(\Psi(x_i, x_j))_{1\leq i,j\leq m}$ is positive-definite for all positive integers m and all mutually distinct $x_1, ..., x_m$.

is 1-1.

The above proposition offers guidance on the choice of kernel in our statistical manifold framework. Throughout the remainder of the thesis, we use the standard and widely used strictly positive-definite kernel function

$$K(u) = \frac{1}{\sqrt{(2\pi)^D}} \exp(-\frac{u^T u}{2}), \tag{6.2.6}$$

with the scaled identity bandwidth matrix, i.e., $\Sigma = \sigma^2 \mathbf{I}$. We note that other choices of kernel function are possible, e.g., the Laplacian kernel, or inverse multiquadratic kernel [87].

From Propositions 6.2.1 and 6.2.2 we have established that, under the distinct points assumption and using the normal kernel function, the mapping $h : \mathcal{X} \to \mathcal{S}$ is 1-1; $\mathcal{S}$ can therefore be given the structure of statistical manifold. Figure 6.2 illustrates statistical manifold representations of some example point clouds.

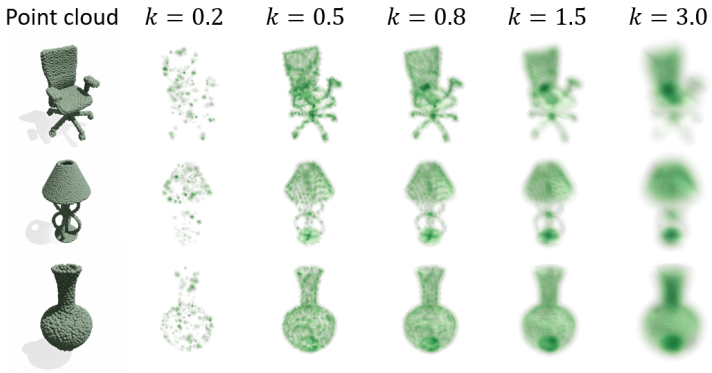


Figure 6.2: Probability heat maps for various k (the greener, the higher) for some examples from the ShapeNet dataset [80], where we set $\sigma = k \times \mathrm{MED}$ for $k \in (0, \infty)$. MED denotes the median of the distances between the points in the point cloud and their nearest points.

6.2.2 Information Riemannian Metric for Point Cloud Data Space

We now equip the point cloud statistical manifold $\mathcal{S}$ with the Fisher information metric, which we refer to as the info-Riemannian metric and denote by $\mathbf{H}$. The first task is to define a local coordinate system on the space of point clouds $\mathcal{X}$. Toward this end,

we use the matrix representation $X \in \mathbb{R}^{n\times D}$ of a point cloud $\mathbf{X}$. Observe that the matrix representation is not unique: given an $n \times n$ permutation matrix $P \in \mathbb{R}^{n\times n}$, then X and PX represent the same point cloud $\mathbf{X}$. Fortunately, this does not cause problems since $p(x; X)$ is defined in a permutation-invariant way, i.e., $p(x; X) = p(x; PX)$ for any $n \times n$ permutation matrix P. We again note that we use italics to denote local coordinate representations, e.g., $\mathbf{X}$ has local coordinates $X \in \mathbb{R}^{n\times D}$, the tangent vector $\mathbf{V} \in T_{\mathbf{X}}\mathcal{X}$ has local coordinates $V \in \mathbb{R}^{n\times D}$.

The info-Riemannian metric $\mathbf{H}$ can be expressed in local coordinates coordinates X as follows:

$$H_{ijkl}(X) := \int p(x; X) \frac{\partial \log p(x; X)}{\partial X^{ij}} \frac{\partial \log p(x; X)}{\partial X^{kl}}\, dx, \tag{6.2.7}$$

for $i, k = 1, ..., n$ and $j, l = 1, ..., D$. Given two tangent vectors $\mathbf{V}, \mathbf{W} \in T_{\mathbf{X}}\mathcal{X}$ with respective matrix representations $V, W \in \mathbb{R}^{n\times D}$, their inner product is then computed as follows:

$$\langle \mathbf{V}, \mathbf{W}\rangle_{\mathbf{X}} := \sum_{i,k=1}^{n} \sum_{j,l=1}^{D} H_{ijkl}(X) V^{ij} W^{kl}. \tag{6.2.8}$$

The coordinate expression of the info-Riemannian metric $H_{ijkl}(X)$ results in a permutation invariant inner product, i.e., $\sum H_{ijkl}(X)V^{ij}W^{kl} = \sum H_{ijkl}(PX)(PV)^{ij}(PW)^{kl}$ for any $n \times n$ permutation matrix P, showing that the metric is geometrically well-defined.

Using the standard normal kernel function, the coordinate expression of the info-Riemannian metric H_{ijkl} has a simple analytic expression as follows:

Proposition 6.2.3. With the standard (multivariate) normal kernel function and the bandwidth parameter σ, the information Riemannian metric $H_{ijkl}(X)$ is given by

$$\int p(x; X) \frac{K(\frac{x-x_i}{\sigma})K(\frac{x-x_k}{\sigma})}{(\sum_{m=1}^{n} K(\frac{x-x_m}{\sigma}))^2} \Big[\frac{(x - x_i)(x - x_k)^T}{\sigma^4}\Big]_{jl}\, dx, \tag{6.2.9}$$

Figure 6.3 shows that, given two moving point cloud data whose velocity matrices have equal Euclidean norm (i.e., $||\mathbf{V}||^2 = \sum_{i=1}^{n}\sum_{j=1}^{D} V^{ij}V^{ij}$), the velocity norms under the info-Riemannian metric are significantly different: the velocity A has a value of 0.2626, while the velocity B has a value of 2.2×10^{-8}. In particular, observe that the tangential velocity in the case B, which does not change the overall distribution of the point cloud, has a very small velocity norm under the info-Riemannian metric as it should.

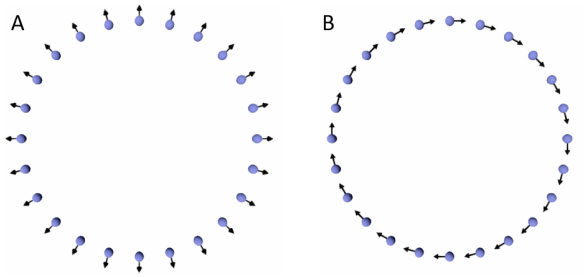


Figure 6.3: Two moving point clouds with different velocity matrices.

6.3 Applications to Point Cloud Autoencoders

Riemannian geometric formulations of autoencoders for representation learning have recently been introduced and extensively studied in [88, 17, 89, 90, 91, 92, 33, 79]. In these works, the image of the decoder function is viewed as a low-dimensional manifold embedded in the high-dimensional data space – we refer to this manifold as the *decoded manifold* – and a Riemannian metric for the decoded manifold is obtained by projecting the data space Riemannian metric to this manifold. In contrast, this perspective cannot be reasonably extended to existing point cloud autoencoders (e.g., FoldingNet [93], AtlasNet [94], AtlasNetV2 [95], and TearingNet [96]), due to the absence of a geometrically well-formulated Riemannian manifold structure.

In this section, using the info-Riemannian metric, we extend this perspective by defining a Riemannian metric for the decoded manifold of the point cloud autoencoder. With this info-Riemannian metric, we examine two case studies: (i) interpolation between two points of latent space via the minimal geodesic; (ii) learning an optimal set of latent space coordinates that best preserves distances and angles (or intuitively, minimizes distortion).

Consider a point cloud decoder function with the m-dimensional latent space $f : \mathbb{R}^m \to \mathbb{R}^{n\times D}$, where the output is expressed in terms of the matrix representation. The projection of the info-Riemannian metric on the point cloud statistical manifold to the decoded manifold is then expressed in latent space coordinates $z \in \mathbb{R}^m$ as follows:

$$G_{ab}(z) := \sum_{i,k=1}^{n} \sum_{j,l=1}^{D} H_{ijkl}(f(z))(J_f)_a^{ij}(z)(J_f)_b^{kl}(z), \tag{6.3.10}$$

where J_f denotes the Jacobian of f, and the indices a, b both range from 1 to m.

The latent space is then assigned a Riemannian metric $G_{ab}(z)$; the following two applications rely on $G(z) \in \mathbb{R}^{m\times m}$.

Geodesic interpolation: The latent space metric $G(z)$ can be used to find the minimal geodesic curve connecting two point clouds (i.e., the shortest length curve in the decoded manifold). Let z_1, z_2 be the encoded values of the two point clouds in the latent space $\mathbb{R}^m$. In terms of the metric $G(z)$, the geodesic curve connecting these two points can be determined as a solution to the following optimization problem [35]:

$$\min_{z(t)} \int_0^1 \dot{z}(t)^T G(z(t))\dot{z}(t)\, dt, \tag{6.3.11}$$

subject to $z(0) = z_1$ and $z(1) = z_2$. Parametrizing $z(t)$ by a cubic spline with fixed boundary points z_1, z_2 then leads to an unconstrained optimization problem. To avoid excessive memory consumption when computing the objective function and its gradient, instead of the usual Riemann sum approximation of the integral, we interpret the integral as an expectation over the uniform distribution $t \sim U(0, 1)$ and accordingly use the mini-batch sampling technique.

Learning optimal latent space coordinates: The latent space metric $G(z)$ can be used to formulate a regularization term when training an autoencoder to learn an optimal set of latent space coordinates; by "optimal" we mean $G(z) = cI$ for some positive scalar c, so that the decoder preserves distances and angles as much as possible. Recently, a regularization technique for this purpose has been introduced in [33]. Specifically, the following regularization term is added to the reconstruction loss function:

$$\mathbb{E}_{z\sim P}[\|G(z) - cI\|_F^2], \tag{6.3.12}$$

where $\|\cdot\|_F$ is the Frobenius norm, $c = \mathbb{E}_{z\sim P}[\frac{1}{m}\mathrm{Tr}(G(z))]$, and P is defined via the modified mix-up augmentation, i.e., $z \sim P \iff z = \alpha z_1 + (1-\alpha)z_2$ where z_1, z_2 are sampled from the set of encoded training data and $\alpha \sim U(-\eta, 1+\eta)$ for $\eta > 0$. For latent spaces whose dimension m is large, in order to avoid the expensive and memory-consuming computation of $G(z) \in \mathbb{R}^{m\times m}$, we use the following regularization term in the subsequent experiments:

$$\mathbb{E}_{z\sim P}[\mathbb{E}_{v\sim\mathcal{N}(0,I)}[\|v^T G(z)v - cv^T v\|^2]], \tag{6.3.13}$$

where we use mini-batch sampling to estimate the expectations. This can be done much more efficiently since we need only compute the Jacobian-vector product, i.e., $\sum_a (J_f)_a^{ij} v^a$.

6.4 Experimental Results

We now verify the effectiveness of the info-Riemannian metric for the two point cloud autoencoder applications described above using both a synthetic 3D basic shape dataset and standard benchmark point cloud datasets.

The synthetic 3D basic shape dataset consists of cylinders, cubes, cones, and ellipsoids with various aspect ratios of the shape parameters (e.g., radius versus height for the cylinder). We sample 512 points from the surface mesh of the shapes using a greedy sample elimination algorithm, so that each sampled point is approximately the same distance from its neighborhood points [97]. Each point cloud is then normalized so that the two farthest points are a unit distance apart. Further details about the synthetic 3D basic shape dataset generation are provided in Appendix D.

The benchmark point cloud dataset consists of ModelNet [81] and ShapeNet [80], where ModelNet consists of ModelNet10 and ModelNet40, each of which consist of 10 and 40 shape classes, respectively. Each point cloud data has 2048 points; we normalize these into a unit sphere as done in [93].

6.4.1 Synthetic 3D Basic Shape Dataset

In Section 6.4.1.1, we use a dataset consisting of cones, cylinders, and ellipsoids, which are split into training/validation/test sets of size 3196/800/804. We then confirm the validity of the proposed metric by comparing the results of several shape interpolation methods in the latent space.

In Section 6.4.1.2, to study the effects of the regularization term when learning the optimal latent coordinates (with respect to the info-Riemannian metric), we use a dataset consisting of boxes, cones, and ellipsoids divided into training/validation/test sets of size 720/240/240.

We use DGCNN as the encoder [98] and a fully-connected neural network as the decoder. The latent space is assumed to be two-dimensional. For the reconstruction loss term, we use the Chamfer distance. The regularization term in Equation (6.3.12) is multiplied by a coefficient $\lambda > 0$ and added to the reconstruction loss term. Further details about the network architectures and training are provided in Appendix D.

6.4.1.1 Example 1: Cone, Cylinder, and Ellipsoid

Figure 6.4 shows the test data encoded in the latent space together with the interpolation trajectories and generated point clouds from those interpolants. In the case of intra-class interpolations (i.e., interpolants between cylinders), the linear interpolant clearly passes through the red ellipsoid region in the latent space, with some of the

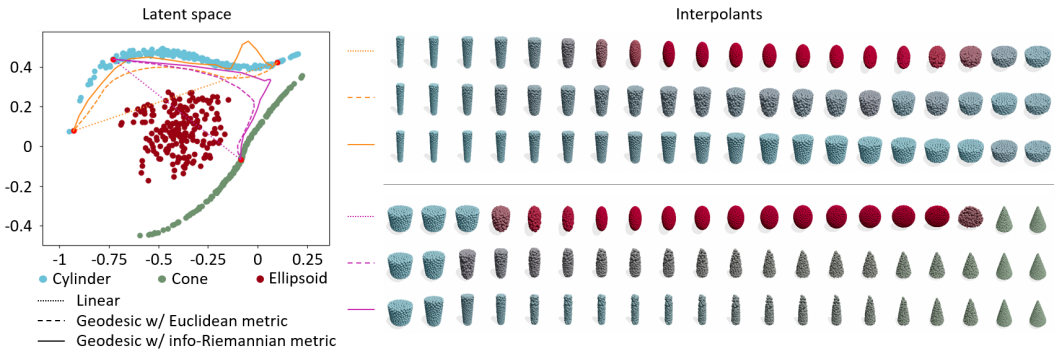


Figure 6.4: *Left*: Latent space with linear and geodesic interpolants. The orange interpolants connect a wide cylinder to a tall cylinder, while the magenta interpolants connect a cylinder to a cone. Linear interpolants and geodesic interpolants under the Euclidean and info-Riemannian metrics are drawn as dotted, dashed, and solid lines, respectively. *Right*: Generated point clouds from those interpolants. To visually indicate which class generated point cloud belong to, we color these according to the ratio of the Chamfer distances to the nearest point cloud for each class (see Appendix D). For example, when it is uncertain which class a generated data belongs to (i.e., the nearest distances to each class are similar), it is assigned some color other than blue, red, or green.

generated point clouds clearly ellipsoids. The geodesic interpolations under the Euclidean and info-Riemannian metrics both avoid ellipsoid regions in the latent space. In particular, the geodesic interpolants between two cylinders are also cylinders; this is well-aligned with human intuition. However, if we look at the generated point clouds in detail, while the info-Riemannian metric produces clearly blue cylinders, some of the generated point clouds with the Euclidean metric are non-blue cylinders (i.e., relatively closer to the ellipsoid region) with noisy side surfaces. For the inter-class interpolations (i.e., interpolants between a cylinder and a cone), the linear interpolant also clearly passes through the red ellipsoid region. The geodesic interpolation under the Euclidean metric produces many non-blue and non-green color shapes during the transition from cylinders to cones, while geodesic interpolation under the info-Riemannian metric produces such cases far less. Overall, it can be observed that the geodesic interpolants under the info-Riemannian metric have minimal shape class changes.

Figure 6.5 shows the latent spaces produced by the regularized autoencoders using the Euclidean and Info-Riemannian metrics. Compared to the latent space in Figure 6.4 (trained without regularization), the following observations can be made: (i)

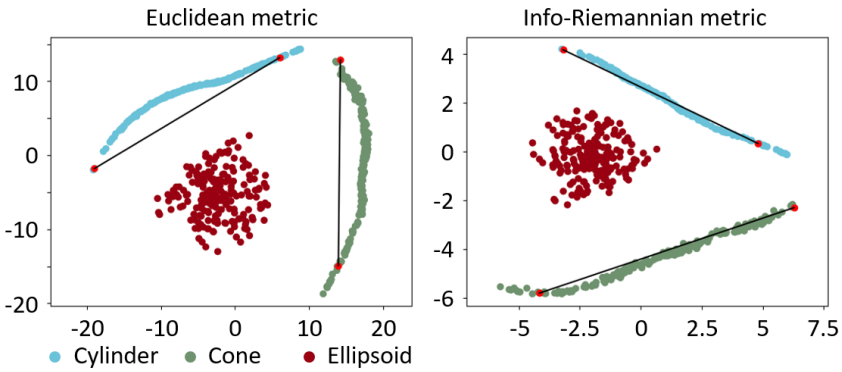


Figure 6.5: Latent spaces produced by regularized autoencoders, each of which is trained with the Euclidean (*Left*) and info-Riemannian metric (*Right*). Representative intra-class linear interpolants between two cylinders and two cones are drawn as black solid lines.

the encoded latent spaces of cones and cylinders are flattened, and (ii) the ellipsoids and cones become more discriminative. Furthermore, we note that the encoded latent space curves of cones and cylinders in the right figure (where the info-Riemannian metric is used) are clearly flatter than those in the left figure (where the Euclidean metric is used). In other words, if we linearly interpolate between two cylinders or two cones, the interpolants in the right case will most likely remain in the same class, unlike the left case (the generated point clouds from the linear interpolants are provided in Appendix D).

6.4.1.2 Example 2: Box, Cone, and Ellipsoid

Figure 6.6 shows the test data encoded in the latent space together with the visualization of the Riemannian metric, the fitted Gaussian Mixture Model and its samples, and pairwise Euclidean distances. From the first column of Figure 6.6, by comparing the results with and without regularization, the following two key observations can be made: (i) in the vanilla autoencoder case (upper case), the major axes of the gray ellipses are aligned with the decision boundary (i.e., a hypersurface that partitions the different class regions), which implies that shapes of different classes are actually more distant in the learned manifold (under the info-Riemmanian metric) than shown in the latent space, and (ii) in the regularized autoencoder case (lower case), by encouraging

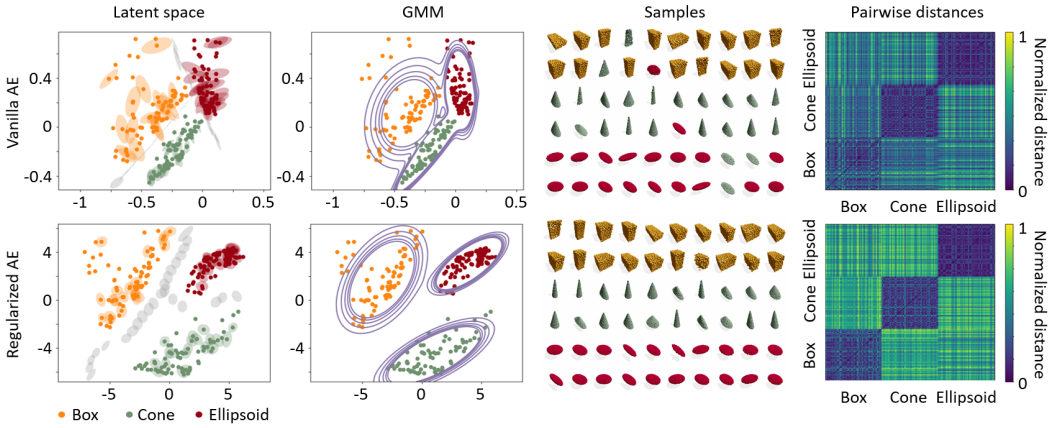


Figure 6.6: From *left* to *right*: latent spaces with equidistant ellipse ($\{z|(z - z^*)^T G(z^*)(z - z^*) = 1\}$ for center z^*) centered on some selected points and sampled points from interspaces, Gaussian Mixture Model (GMM) fitting results, generated samples from the GMM, and the heat map of the pairwise Euclidean distances in the latent space of all test data. The *upper* figure is a vanilla autoencoder trained without regularization, while the *lower* figure is trained with regularization (using the info-Riemannian metric). For the samples in the third column, we assign colors using the same method of Section 6.4.1.1 to visually express which classes the samples are likely to belong to.

the metric to be isotropic (i.e., turning ellipses into circles), the gaps between different class regions are widened. The second column confirms that the components of the GMM are better separated after regularization; each component of the GMM on the regularized autoencoder generates high-quality, even samples from the same class shape as shown in the third column. The heat maps of the pairwise distances in the last column also indicate that shapes in different classes are more distant, and therefore more easily separable in the latent space of the regularized autoencoder.

To quantitatively verify, we repeat this experiment with multiple different synthetic datasets (details are in Appendix D), and report the averaged GMM clustering scores, Normalized Mutual Information (NMI) and Adjusted Rand Index (ADI), in Table 6.1. Ours shows higher clustering accuracy.

Table 6.1: Averaged clustering scores over 27 different datasets, each of which consists of diverse shapes of boxes, cones, and ellipsoids; the higher the better.

METHOD	NMI	ADI
Vanilla AE	0.7624 $\pm$ 0.2132	0.7209 $\pm$ 0.2598
Regularized AE	**0.9484** $\pm$ **0.1391**	**0.9368** $\pm$ **0.1737**

6.4.2 Standard Benchmark Data

To show that the regularization technique with the info-Riemannian metric can benefit unsupervised representation learning from the perspective of discriminative representation learning, we compare the transfer classification accuracy of ShapeNetCore.v2 to ModelNet following the same experimental procedure outlined in [93]. When training autoencoders with ShapeNet, random rotations about an axis parallel to the direction of gravity are applied to each point cloud. We use four different point cloud autoencoders: *FcNet* and *FoldingNet* adopted from [93], *PointCapsNet* adopted from [99], and *DGCNN-FcNet* using DGCNN [98]; the latent space is 512-dimensional for all. The four autoencoders are trained with and without regularization.

In the former case, the regularization terms of Equation (6.3.13) under both the Euclidean and info-Riemannian metrics are used while varying the regularization coefficient λ. We distinguish between regularized autoencoders using the Euclidean and info-Riemannian metrics by an "+E" or "+I" after the network name. After network training is finished, we train linear SVM classifiers with the encoded data for ModelNet10 and ModelNet40. These are split into training/test sets of sizes 3991/909 and 9843/2468, respectively. Further experimental detail are provided in Appendix D.

Table 6.2 shows a comparison of transfer classification accuracy from ShapeNet to ModelNet10 (MN10) and ModelNet40 (MN40) for various recent state-of-the-art methods. In the upper table (*Adopted from References*), the numbers are adopted from previous papers (the experimental procedures may differ slightly from ours). In the lower table (*Implemented by Authors*), we report the best numbers obtained. First, although their performance is not directly comparable due to differences in the experimental procedures, it can be seen that our regularized autoencoders are comparable to the state-of-the-art methods. Second, at least for our implementation, regularization using the info-Riemannian metric improves classification accuracy over vanilla autoencoders, with higher accuracy compared to the Euclidean metric case.

Next, based on the intuition that kernel-based statistical representations are robust to noise, we also conduct additional experiments to determine how much more robust the representation obtained with our regularization approach is for noisy point cloud

Table 6.2: Classification accuracy by transfer learning for ModelNet10 (MN10) and ModelNet40 (MN40) from ShapeNet.

METHOD	MN40	MN10
Adopted from References		
SPH [100]	68.2%	79.8%
LFD [101]	75.5%	79.9%
VConv-DAE [102]	75.5%	80.5%
3D-GAN [103]	83.3%	91.0%
Latent-GAN [104]	84.5%	**95.4%**
FoldingNet [93]	88.4%	94.4%
PointFlow [105]	86.8%	93.7%
Multi-Task [106]	89.1%	-
PointCapsNet [99]	**89.3%**	-
Implemented by Authors		
FcNet	88.3%	93.5%
FcNet + E (ours)	89.3%	93.7%
FcNet + I (ours)	90.4%	94.3%
FoldingNet	89.3%	93.7%
FoldingNet + E (ours)	88.9%	94.4%
FoldingNet + I (ours)	90.1%	94.5%
PointCapsNet	87.2%	93.6%
PointCapsNet + E (ours)	88.1%	93.7%
PointCapsNet + I (ours)	88.5%	93.9%
DGCNN-FcNet	90.3%	94.5%
DGCNN-FcNet + E (ours)	89.9%	94.4%
DGCNN-FcNet + I (ours)	**91.0%**	**95.2%**

data. We add noise with different levels of standard deviation (1%, 5%, 10%, and 20% of the diagonal length of the point cloud bounding box) to point cloud data (see Appendix D for details). Then *FcNet* is trained with and without regularization in the same way as above.

Table 6.3 shows a comparison of transfer classification accuracy in the presence of noise. As the noise level increases, the classification accuracy obviously decreases, but the reduction is the least dramatic for regularized autoencoders under Info-Riemannian metric. Figure 6.7 shows learning curves, ModelNet40 transfer classification accuracy and reconstruction error as functions of the training epoch, in the presence of noise. Throughout the learning process, compared to vanilla autoencoders (light colored lines), regularized autoencoders under Info-Riemannian metric (dark colored lines) show higher classification accuracy and similar levels of reconstruction errors. In particular, as shown in the right figure, as the learning progresses, the classification accuracy of the vanilla autoencoder largely decreases in expense of minimizing the reconstruction error, while

that of our regularized autoencoder is maintained. This shows our regularization approach helps autoencoders to learn robust representations to noise as expected.

Table 6.3: Classification accuracy by transfer learning for ModelNet10 (MN10) and ModelNet40 (MN40) from ShapeNet under the noise levels of 1%, 5%, 10%, and 20%.

	MN40			
METHOD	1%	5%	10%	20%
FcNet	87.8%	83.2%	75.6%	64.5%
FcNet + E (ours)	86.6%	85.1%	79.1%	70.4%
FcNet + I (ours)	**89.0%**	**86.6%**	**81.4%**	**72.4%**
	MN10			
	1%	5%	10%	20%
FcNet	92.4%	91.9%	88.4%	79.8%
FcNet + E (ours)	92.2%	91.1%	88.2%	82.6%
FcNet + I (ours)	**93.3%**	**92.6%**	**91.6%**	**84.8%**

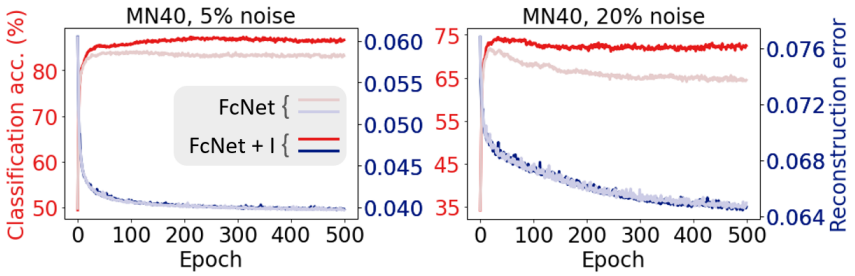


Figure 6.7: Learning curves in the presence of noise (*left*: 5% noise; *right*: 20% noise), ModelNet40 transfer classification accuracy and reconstruction error as functions of the training epoch.

Lastly, we compare the semi-supervised transfer classification accuracy. In the semi-supervised settings, not all the training data have labels, which are actually more frequent situations in reality. When we train linear SVM classifier, we use different numbers of training data (1%, 5%, 10%, and 50% of the overall training data, see Appendix D for details). Table 6.4 shows a comparison of transfer classification accuracy according to the percentage of training label used. As the label rate decreases, the classification accuracy decreases, but the reduction is more dramatic for vanilla autoencoders. Together with the results in Table 6.2, this clearly shows that our regularization approach helps autoencoders to learn more discriminative representation spaces, and the effect is greater with a small number of labeled data.

Table 6.4: Classification accuracy by transfer learning for ModelNet10 (MN10) and ModelNet40 (MN40) from ShapeNet under the different percentages of labeled training data for linear SVM classifier (50%, 10%, 5%, and 1%).

METHOD	MN40			
	50%	10%	5%	1%
FcNet	85.7%	78.0%	70.6%	50.3%
FcNet + I (ours)	**87.9%**	**81.6%**	**76.8%**	**57.4%**
	MN10			
	50%	10%	5%	1%
FcNet	91.7%	90.1%	87.2%	74.1%
FcNet + I (ours)	**93.2%**	**91.2%**	**88.3%**	**78.1%**

Overall, it is indeed somewhat surprising that unsupervised classification accuracy can be improved (i.e., more discriminative representation space can be obtained) with a simple regularization technique in lieu of a complex neural network architecture or loss function. We include additional experimental results and analysis in Appendix D.

6.5 Discussion and Conclusion

We have proposed a new Riemannian geometric structure for the space of point cloud data. We have defined a statistical representation of point cloud data and constructed a statistical manifold in a mathematically rigorous way. Then a natural Riemannian metric – Fisher information metric – is assigned to the point cloud statistical manifold, which provides geometrically well-defined measures needed for applications. We demonstrate its advantages through two applications involving point cloud autoencoders: (i) minimal geodesic interpolants under info-Riemannian metric have minimal shape changes compared to the standard linear interpolants, and (ii) the optimal latent coordinates learned using our method produce more discriminative representation spaces than existing methods. In particular, transfer classification accuracy has been greatly improved in noisy data and semi-supervised settings.

As a potential issue, the "fixed number of points" assumption used in our construction of the statistical manifold may be violated in real world problems. In such cases, we can easily mitigate this issue by matching the number of points in each point cloud through a simple upsampling/downsampling algorithm. Further, the kernel function used in our current implementations, the standard normal kernel function, may not be an optimal choice. Other choices can be explored to enhance our algorithms as long as the conditions of Proposition 6.2.2 are satisfied.

Compared to research on distance metric, there are relatively few studies on Riemannian metric despite its importance and utility. This work is the starting point of research on point cloud space Riemannian metric, and we believe the study on diverse Riemannian metrics, just as various distance metrics with different properties have been developed, should be continued.

In certain real-life scenarios containing multiple 3d objects that (i) are only partially observed (e.g., only one side of the underlying surface is observed), and (ii) local densities of the measured points for each object are different, using a single probability density function with a single kernel-type and fixed bandwidth parameter to represent the measured point cloud can be problematic. For example, consider a point cloud data obtained through LiDAR; the point cloud can include diverse objects such as cars, pedestrians, trees, buildings, and lanes. They are obviously partially observed, and local point cloud densities are different since their distances from the sensors are different. One way to approach this problem is to (i) first decompose the measured point cloud into several multiple point clouds, each of which represents a single object (for example, by using existing object detection techniques); (ii) use point cloud completion and super resolution algorithms as needed to make each point cloud rich enough to represent the corresponding object, and (iii) apply our methods by using different kernels and bandwidth parameters suitable to represent each point cloud. Other approaches are also possible, and we believe it is an worthwhile future research topic to examine which approaches are best suited to different application domains.

7 Conclusion

7.1 Summary and Discussion

This thesis has proposed geometric methods that solve the two problems of the vanilla autoencoders introduced in Chapter 1.3: (i) wrong manifold and (ii) distorted latent space problems. The Neighborhood Reconstructing Autoencoder in Chapter 3 and Minimum Curvature Autoencoder in Chapter 4 are solutions to "the wrong manifold problem", where the former exploits the neighborhood graph information constructed in the data space and the later is based on the minimum extrinsic curvature principle. The Isometric Regularization in Chapter 5 is the solution to "the distorted latent space problem", and its one application case with point cloud data is introduced in Chapter 6.

Contrary to many existing autoencoder regularization methods that focus on regularizing the latent space distributions that are entirely determined by the encoders, we have discovered that the decoders play equally or sometimes more important roles than the encoders in autoencoder-based manifold representation learning. This is because the decoder explicitly parametrizes the manifold hence the decoder is aware of how the manifold actually lies in the data space while the encoder is not, and local geometry of the manifold can be described by taking higher-order derivatives of the decoder.

The proposed algorithms can be classified based on what needs to be determined in prior to use the algorithms. The neighborhood reconstructing autoencoder needs a prior construction of the **neighborhood graph**, whereas the minimum curvature autoencoder and isometric regularization method need to define the **Riemannian metric** for the ambient data space (Figure 7.1). In particular, the curvature and distortion measure have been carefully defined to be **coordinate-invariant** so that they measure geometrically meaningful quantities. Apparently, the proposed algorithms depend on either the graph or Riemannian metric $H(x)$ construction. In this thesis, we have

explored one of the simplest choices: the k-nn graph construction method and the identity metric for $H(x)$. It is notable that, even with these simple choices, our algorithms outperform most existing autoencoder regularization methods.

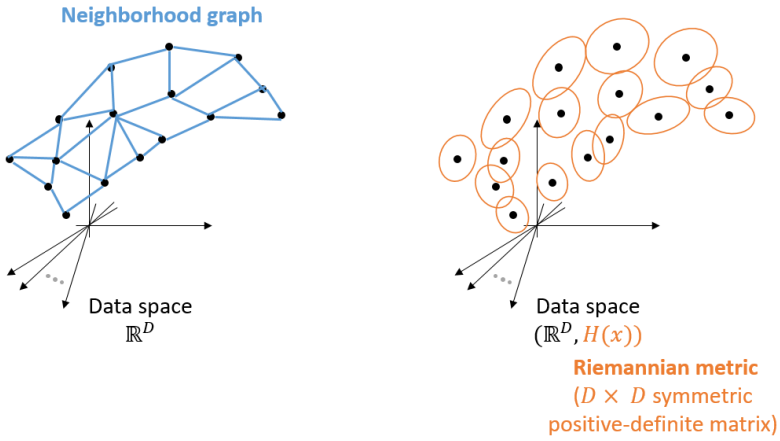


Figure 7.1: The proposed algorithms either require to construct a neighborhood graph or Riemannian metric for the data space.

It is particularly worthwhile to compare the Neighborhood Reconstructing Autoencoder (NRAE) and Minimum Curvature Autoencoder (MCAE). Recall the autoencoder's manifold estimation perspective discussed in the previous Chapter 1.4. Interestingly, the NRAE asymptotically converges to the vanilla autoencoder – since the neighborhood graph $\mathcal{N}(x)$ converges to the center point x –, and thus is an asymptotically unbiased manifold estimator just as the vanilla autoencoder. On the other hand, the MCAE designed to learn a flatter manifold is biased unless the ground truth manifold is flat. Nevertheless, in non-asymptotic situations where the number of training data is limited, the inductive biases that NRAE and MCAE have can help learning accurate manifolds. In NRAE, the manifold is trained to be locally linear or quadratic within a region specified by the local neighborhood graph. In MCAE, the manifold is trained to be flat. Both approaches can be seen as learning smooth manifolds. While the NRAE smooth out the manifold considering local density of data and local directions that data lie, the MCAE smooth out the manifold with the same local weights in all directions and tends to produce overly-flattened manifolds. In comparison, NRAE generally has higher manifold learning performance than MCAE. Meanwhile, we have empirically observed that the MCAE shows strong robustness to noise, which seems to be because the manifold fitted to noisy data tends to be curvier than the ground truth manifold.

We have conducted a wide range experiments with image, motion capture, and point cloud data sets, compared to existing state-of-the-art methods, and shown that our methods learn more accurate manifold and its representation with less distortion, improving performance for standard downstream tasks such as image retrieval, clustering, and classification, in some cases by significant margins.

7.2 Limitations and Future Directions

Finding good lower-dimensional representations of real-world complex high-dimensional problems – for example, in the presence of multiple objects not aligned in the center of the image, bias in the training data set (e.g., the background of images containing birds is highly likely the sky), data corruptions by outliers, noises, and occlusions, and many others – is still a challenging problem, and our algorithms have the potential to be further improved in many directions. In this section, we pose some limitations of the current implementations of our manifold representation learning algorithms and discuss some promising future research directions. The rest of the section is organized as follows: (i) Graph and Riemannian Metric Construction, (ii) Manifold with Complex Topology, and (iii) Disentangled Representation.

7.2.1 Graph and Riemannian Metric Construction

In the current implementations of our algorithms in Chapters 3, 4, 5, we have only considered unsupervised settings and assumed Euclidean geometry of data spaces when constructing the graph or Riemannian metric. However, in real-world problems, Euclidean geometry may not reflect the geometric structures of data manifolds that we expect and may not be optimal for the downstream tasks (just as shown in Chapter 6 with point cloud data).

For example, suppose we want to train a machine that plays a maze game, and the set of maze game screen images is given as a training data set for representation learning; see Figure 7.2. Clearly, the screen image data form a two-dimensional manifold since the position of an agent, i.e., the red circle, is fully described by two variables, i.e., xy-position. Considering the characteristics of the maze game, the distance between images (a) and (b) should be closer than the distance between images (a) and (c) in the learned representation space. However, a graph or Riemannian metric constructed based on the Euclidean geometry of the image space does not capture the geometry of the maze structure.

As another example, suppose we want to find lower-dimensional representations of

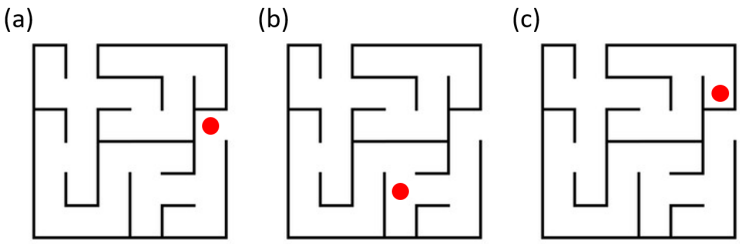


Figure 7.2: Maze Game Example.

cat and dog images suitable for the classification task. If we assume Euclidean geometry of the image space, even in the lower-dimensional cat and dog image manifold, the geodesic distance between the cat and dog images in Figure 7.3(a) and Figure 7.3(b) is closer than the geodesic distance between the dog and dog images in Figure 7.3(b) and Figure 7.3(c), since the faces are aligned so well in the images (a) and (b). Therefore, dog and cat images will not be well separated in the learned manifold representation space.

Figure 7.3: Cat and dog images.

As such, unsupervised learning approaches have clear limitations since the canonical Euclidean geometry of data spaces does not always capture the geometric structures optimal for downstream tasks. A promising next step would be to exploit supervised signals (e.g., labels, self-supervision via data augmentations, interactions with environments, temporal relations in time series data) for graph and Riemannian metric constructions; then our autoencoder methods can be naturally used without any significant modifications. For example, in reinforcement learning settings, high-dimensional observation data are usually given as a sequential data; we may construct graphs or Riemannian metrics that capture the temporal proximity of data. Or, for classification tasks, class labels or class-preserving data augmentation techniques may be used to construct graphs or Riemannian metrics that capture class-aware geometric structures.

There are unsupervised local metric learning and supervised local metric learning literatures either non-parametric or parametric, which are closely related to our isometric representation learning methods [107, 108, 109, 110, 92].

7.2.2 Manifold with Complex Topology

In the current implementations of our autoencoder methods, we use a single Cartesian space $\mathbb{R}^m$ as the latent space, which implicitly assumes that the data manifold is homeomorphic to $\mathbb{R}^m$. However, in real-world scenarios, this assumption mostly does not hold. Data manifolds can be in fact homeomorphic to $S^1, S^2, \mathrm{SO}(3)$ [111, 112], or more complex topological manifolds, or may have locally different dimensions [113]. If we could know the topology of the data manifold in prior, we would be able to define a homeomorphic latent space, however, in most cases, it is unknown.

For example, real world images that include multiple objects with various background scenes less likely lie on smooth manifold of constant dimension that is homeomorphic to the Cartesian space. For each image, the number of objects varies, the dimension and topology for each object manifold is different, and, when objects are overlapped in the image, it is difficult to imagine how geometrically the object manifolds would intersect in the image data space. It is very challenging to identify an arbitrary complex lower-dimensional structure and the corresponding representation.

Reluctantly but compelled, current existing approaches including ours select a big enough latent space dimension, so that the complex geometric structure can be embedded in a higher dimensional latent space. As a result, the image of the latent space does not exactly match the data manifold, rather the data manifold is included in the image of the latent space, and some latent points do not produce meaningful data points (e.g., in the image manifold representation learning, unnatural and highly distorted images can be generated).

There could be two possible future research directions: (i) using multiple coordinate charts [114] and (ii) re-identifying geometric structure of the data distribution in the latent space. Multiple coordinate charts may be useful for describing complex data manifold of non-constant dimension, i.e., multiple decoders each of which spans only part of the data manifold. In this approach, another challenge arises in that we need to identify the intersections between the multiple charts. Or, we may identify geometric structures (e.g., manifold) again in the latent space. The key difference to the original problem is the dimensionality: since the latent space dimension is much lower-dimensional than the original data space, machine learning methods difficult to use in high-dimensional spaces, e.g., non-parametric methods, are expected to be feasible to use.

7.2.3 Disentangled Representation

Assuming high-dimensional observations are in fact a manifestation of a low-dimensional set of independent ground-truth *factors of variation* – for instance, in human face data, the size of eyes, noses, and the hair color are examples of the factors of variation –, the ability to disentangle these factors has been a critical ingredient in *disentangled representation learning* [115]. This abstract concept has spawned various formalizations, but the community has yet to agree on a single definition of disentangled representation.

The most popular one is the probabilistic definition based on the (statistical) independence prior factorization [53, 116, 117, 118]. However, as reported in recent studies [119, 120], this definition is severely under-specified and does not reflect the commonly used intuitive notion of disentanglement, which is often referred to as the "impossibility result". On the other hand, there are recently introduced geometric definitions that either rely on the group theory or product manifold assumption [121, 122], which intuitively better reflect our abstract sense of disentanglement. However, these definitions have many and complex elements to identify such as the groups, group actions, projection mappings for sub-manifolds [123, 124, 122].

We believe our geometric regularization methods are closely linked to the disentangled representation learning. First of all, learning a correct manifold, or more specifically a correctly-connected manifold, is a necessary condition to learn the disentangled representation. For example, consider a two-dimensional cat image data manifold with different fur colors and poses and its disentangled representation as shown in Figure 7.4 (*Left*). In this disentangled latent coordinates, translation on one axis changes only the cat's pose, and translation on the other axis changes only the fur color. However, vanilla autoencoder does learn this disentangled representation as shown in Figure 7.4 (*Right*). One of the obvious problems is that the vanilla AE has learned an incorrectly-connected data manifold as visualized in the red box, where abrupt changes in images occur. Learned data manifolds should be accurate – in this case, smoothly connected – so as to have disentangled representations where smooth transitions in the latent spaces produce smooth changes in data.

Secondly, we view that the disentanglement property is closely related to the geometry-preserving property of the decoder. Suppose we have a data point $x \in \mathbb{R}^D$ in an m-dimensional data manifold $\mathcal{M} \subset \mathbb{R}^D$ ($D \geq m$), and a set of m infinitesimal direction vectors $e_i(x) \in \mathbb{R}^D, i = 1, \ldots, m$ that are disentangled, i.e., each i-th change $x \mapsto x + e_i(x)$ only changes one factor. If we could find a coordinate system $f : \mathbb{R}^m \to \mathcal{M}$ such that $x = f(z)$ and $f(z + dz^i) = x + e_i(x)$ for all i, z, then the coordinates can be considered as the disentangled representation as illustrated in Figure 7.5. The condition can be written as $J_f(z)dz^i = e_i(f(z))$. Assuming $\|dz\| = 1$ for simplicity, then $\left(J_f^T(z)J_f(z)\right)_{ij} = e_i(f(z))^T e_j(f(z))$. This condition is an equality constraint on the

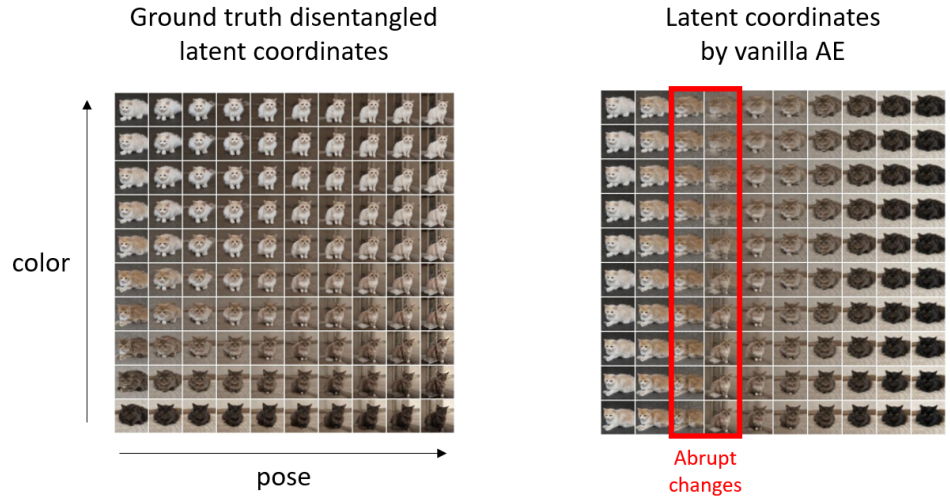


Figure 7.4: *Left*: An example of the disentangle representation. *Right*: Vanilla AE learns an entangled representation. It is adopted from [125].

space of Riemannian metrics (i.e., $m \times m$ positive-definite tensor fields), and it would be interesting to see if we can construct a Riemannian metric $H(x)$ for $\mathbb{R}^D$ by using e_i so that the aforementioned equality condition becomes identical to the isometry condition.

7.3 Concluding Remark

A class of geometric methods proposed in this thesis can be used to learn lower-dimensional manifold representations for high-dimensional data. The next step is to explore more advanced structures in the manifold representation space. For example, it is desired to know what changes in a particular direction in the representation space physically mean, e.g., interpretable and disentangled representation, and how interactions with environments can be described mathematically in the representation space, e.g., group action on a manifold. Investigating the manifold representation space for more advanced geometric structures will be important future research directions.

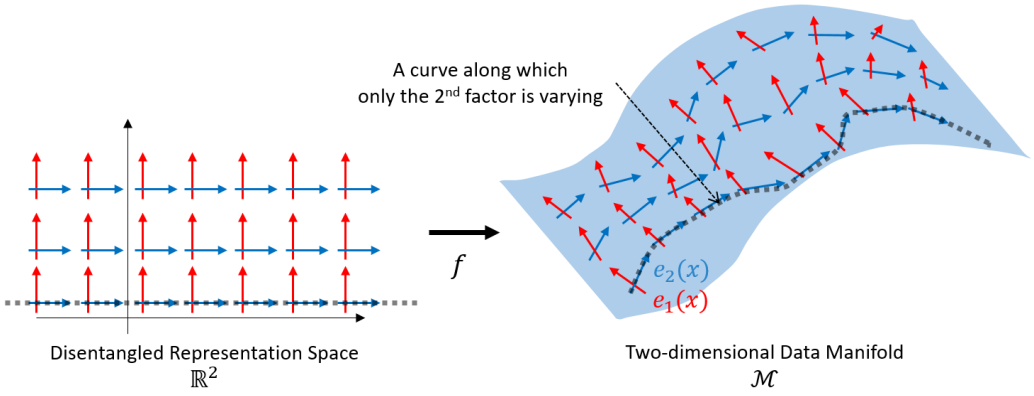


Figure 7.5: Disentangling by isometry.

A

Appendix: Neighborhood Reconstructing Autoencoders

A.1 Experimental Details

In what follows we will call each experiment by its corresponding figure or table number for convenience.

A.1.1 Dataset

Except for a few synthetic data whose generation processes are described in the main script (Figure 3.4, 3.7), we use the standard benchmark datasets downloaded from TorchVision library. For the rotated/shifted MNIST images (Figure 3.8, 3.9), we use the Affine transformation function in the TorchVision library.

For Figure 3.5, we select the first 100 images of digit 8 from 50000 training data in the original MNIST dataset and rotate by 3.6 degrees 100 times to generate a new 10000 training data. For Figure 3.6, we use the entire 50000 training data for training. For the rotated MNIST of digit 3 in Figure 3.8, we select the first image of digit 3 from the original MNIST dataset and rotate by 9 degrees 20 times to generate a new 20 training data. For the shifted MNIST of digit 7 in Figure 3.8, we select the first image of digit 7 from the original MNIST dataset and transform with scale 0.8 and shift range [-10,10] to generate 20 training data. For Figure 3.9, we select the first image of digit 6 from the original MNIST dataset and rotate by 9 degrees 20 times to generate a new 20 training data. For Figure 3.10, we use the 1000,2000,...,10000 training data selected from the training dataset, 10000 validation data, and 50000 test data.

In experiments (Table 3.2, 3.3, 3.4, 3.5), we use either or both of the Large (L)

and Small (S) dataset for the standard benchmark vision data: MNIST, FMNIST, KMNIST, Omniglot, SVHN, CIFAR10, CIFAR100, CELEBA. The large denotes the use of entire public data where training, validation, and test splits are (50000,10000,10000) for MNIST, FMNIST, KMNIST, (15000,4280,13180) for Omnigolot, (60000, 13257, 26032) for SVHN, (45000,5000,10000) for CIFAR10, CIFAR100, and (162770,19867,19962) for CELEBA. The small denotes the use of 20 to 30 percents of the entire public training data where training, validation, and test splits are (10000,2000,50000) for MNIST, FMNIST, KMNIST, SVHN, CIFAR10, CIFAR100, (8000,1000,4180) for Omnigolot, and (50000,10000,100000) for CELEBA. For Table 3.2, we use the large dataset. For Table 3.3, we use both large and small datasets. For Table 3.4 and 5, we use the small dataset.

A.1.2 Network Architecture

In this thesis, we use fully connected neural network and convolutional neural network. For VAE, we use the Gaussian encoders whose output dimension is always doubled to represent both mean and variance, and use the isotropic Gaussian decoders with trainable variances. For DAE, we use the Gaussian noise in training. For WAE, we use the MMD loss and median heuristic.

Fully connected neural network (Figure 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 3.10) For Figure 3.4, we use the networks of size (2-1024-1024-1) and (1-1024-1024-2) with ReLU activation functions for the encoder and decoder, respectively. For Figure 3.5, we use the networks of size (784-1024-1024-1024-2) and (2-1024-1024-1024-784) with ReLU activation functions for the encoder and decoder, respectively. For Figure 3.6, we use the networks of size (784-500-500-2) and (2-500-500-784) with Softplus activation functions (a smooth approximation of the ReLU function) for the encoder and decoder, respectively. For Figure 3.7, we use the networks of size (2-512-512-1) and (1-512-512-2) with ReLU activation functions for the encoder and decoder, respectively. For the rotated MNIST of digit 3 in Figure 3.8, we use the networks of size (784-32-32-1) and (1-32-32-784) with ReLU activation functions for the encoder and decoder, respectively. For the shifted MNIST of digit 7 in Figure 3.8, we use the networks of size (784-128-128-128-128-1) and (1-128-128-128-128-784) with ReLU activation functions for the encoder and decoder, respectively. For Figure 3.9, we use the networks of size (784-512-512-2) and (2-512-512-784) with ReLU activation functions for the encoder and decoder, respectively, where the latent values are normalized after encoding. For Figure 3.10, we use the networks of size (784-512-512-m) and (m-512-512-784) with ELU activation functions for the encoder and decoder, respectively. m is 16, 32, and

32 for MNIST, FMNIST, and KMNIST, respectively.

Convolutional neural network (Table 3.2, 3.3, 3.4, 3.5) We will denote a convolution and deconvolution layer by Conv2d(a,b,c,d,e) and ConvTranspose2d(a,b,c,d,e), respectively, where a is the number of input channel, b is the number of output channel, c is the kernel size, d is the stride (biases are always set True), and e is the padding. We use ReLU activation functions.

For MNIST, FMNIST, KMNIST, Omniglot whose image sizes are (1,28,28), we use the convolutional encoder networks

$$\text{Conv2d}(1,128,3,2,0) \text{ - } \text{Conv2d}(128,256,3,2,0) \text{ - } \text{Conv2d}(256,512,3,2,0) \text{ - } \text{Conv2d}(512,1024,3,2,0) \text{ - } \text{Conv2d}(1024,z_{\text{dim}},1,1,0),$$

and the decoder networks

$$\text{ConvTranspose2d}(z_{\text{dim}},1024,8,1,0) \text{ - } \text{ConvTranspose2d}(1024,512,3,2,1) \text{ - } \text{ConvTranspose2d}(512,256,2,2,1) \text{ - } \text{ConvTranspose2d}(256,1,1,1,0),$$

where z_{dim} is 16,32,32,32, respectively.

For SVHN, CIFAR10, CFIAR100 whose image sizes are (3,32,32), we use the convolutional encoder networks

$$\text{Conv2d}(3,128,4,2,0) \text{ - } \text{Conv2d}(128,256,4,2,0) \text{ - } \text{Conv2d}(256,512,4,2,0) \text{ - } \text{Conv2d}(512,1024,2,2,0) \text{ - } \text{Conv2d}(1024,z_{\text{dim}},1,1,0),$$

and the decoder networks

$$\text{ConvTranspose2d}(z_{\text{dim}},1024,8,1,0) \text{ - } \text{ConvTranspose2d}(1024,512,4,2,1) \text{ - } \text{ConvTranspose2d}(512,256,4,2,1) \text{ - } \text{ConvTranspose2d}(256,3,1,1,0),$$

where z_{dim} is 64,128,128, respectively.

For CELEBA whose image size is (3,64,64), we use the convolutional encoder networks

$$\text{Conv2d}(3,128,5,2,0) \text{ - } \text{Conv2d}(128,256,5,2,0) \text{ - } \text{Conv2d}(256,512,5,2,0) \text{ - } \text{Conv2d}(512,1024,5,2,0) \text{ - } \text{Conv2d}(1024,z_{\text{dim}},1,1,0),$$

and the decoder networks

$$\text{ConvTranspose2d}(z_{\text{dim}},1024,8,1,0) \text{ - } \text{ConvTranspose2d}(1024,512,4,2,1) \text{ - } \text{ConvTranspose2d}(512,256,4,2,1) \text{ - } \text{ConvTranspose2d}(256,128,4,2,1) \text{ - } \text{ConvTranspose2d}(128,3,1,1,0),$$

where z_{dim} is 128.

A.1.3 Training Details

We first introduce how we choose the hyperparameter for each algorithm and experiment. AE, VAE do not have hyperparameters, WAE, CAE, SPAE have the regularization coefficients, DAE has the noise level, and NRAE-L, NRAE-Q have λ. For NRAE, we fix the number of nearest neighbors as 15 for MNIST, FMNIST, KMNIST, Omniglot, SVHN, CELEBA (S) and 30 for CIFAR10, CIFAR100, CELEBA (L). We note that these values are not found through extensive search processes, nevertheless, our algorithms show good results. The same numbers of nearest neighbors are used for SPAE except for the CELEBA where we choose 15 nearest neighbors because the computational complexity of SPAE largely increases as the number of nearest neighbors increases.

For experiments that show qualitative results (Figure 3.4, 3.5, 3.6, 3.7, 3.8, 3.9), we try our best for searching proper hyperparameters for all experiments. For experiments that show quantitative results, we select the best hyperparameters based on the reconstruction error metrics evaluated with the validation data. For Figure 3.10, Table 3.3, the regularization coefficients for CAE, WAE are searched around $0.01 \sim 0.001$, the noise level used in DAE is searched around $0.1 \sim 0.01$, and the regularization coefficient and λ for SPAE and NRAE are searched around $0.001 \sim 0.0001$. For Table 3.2, we use the best hyperparameters selected from the Table 3.3, 3.5 except for the DAE (we use the same noise statistics for DAE in training as the added noise statistics). For Table 3.4, we search the parameters over the joint hyperparameter spaces.

For Figure 3.10 and Table 3.3, 3.4, 3.5, we use the Adam optimizer with learning rates $0.001 \sim 0.0001$ for 1000 epochs using 100 batch size. We use the following early-stopping criteria: we stop the training if the mean reconstruction error on validation data increases 10 times in a row. For Table 3.2, we did not use the early-stopping since we do not have clean data in this setting.

In experiments, we use the following GPU: TITAN X (Pascal), GeForce GRX 1080 Ti, GeForce RTX 2080 Ti, GeForce RTX 3090, each of which RAM has $10 \sim 24$ GB memory. For CAE, we need 24 GB RAM. We did not use multiple GPUs for each experiment; a single machine is enough to run the experiments.

A.2 Additional Experimental Results.

A.2.1 Computational Time

In this section, all experiments are performed on TITAN X (Pascal) with 12GB RAM. We first compare the per-epoch runtime of NRAE with the vanilla AE, VAE and the graph-based method TopoAE in Table A.1. The runtime of TopoAE is adopted from

the original paper, and we take experiments of AE, VAE, and NRAE with the same setting (100 batch size with MNIST dataset, 3 hidden FC layers). Although the device and other environments used in TopoAE experiment can be different from those used in our experiments, as shown in Table A.1, the difference in computational time is big enough to compensate those differences in devices and environments. As discussed in the main script, TopoAE-like methods that require to compute the topological features using the persistent homology at every training iteration are yet computationally very expensive.

Table A.1: Comparisons of the per-epoch runtime with 100 batch size with MNIST dataset. The network architecture is composed of 3 hidden FC layers (1000-500-250) and (250-500-1000) for the encoder and decoder, respectively, with two-dimensional latent space (The runtime of TopoAE is adopted from the original paper).

	AE	VAE	NRAE-L	NRAE-Q	TopoAE
time (s)	1.74	2.17	3.13	4.15	68

Using the fully connected neural networks, we compute the per-epoch runtimes (100 batch size and 50000 training data) of our algorithms and the baselines (Figure A.1). Firstly, we compute the per-epoch runtimes by changing the input dimension as 100, 500, 1000, 3000, 5000. The runtime of CAE rapidly increases because it uses the full Jacobian in the loss function, and, when the input dimension is beyond 1000, we couldn't run the experiments due to the GPU memory limitation. On the other hand, the runtimes of our algorithms are comparable with other existing methods. Secondly, we compare the runtimes of our algorithms with the graph-based method SPAE by changing the number of nearest neighborhood points. The runtime of the SPAE linearly increases (logarithmic in the graph) since it requires to compute the forward pass of the encoder function as many times as the number of neighborhood points during training. In contrast, our algorithms can use the batch sampling method for the neighborhood points, thus the runtimes maintain constant.

A.2.2 Robustness to the Choice of the Number of Nearest Neighbors

It is reasonable to ask how robust the NARE is to the choice of the number of nearest neighbors. We take an experiment with the MNIST dataset to show its behavior given varying number of nearest neighbors. Figure A.2 shows the test mean reconstruction errors of NRAE as a function of the number of nearest neighbors compared to the other baselines. As shown in the figure, the generalization performances of NRAE are mostly better than the other baselines robustly to the number of nearest neighbors.

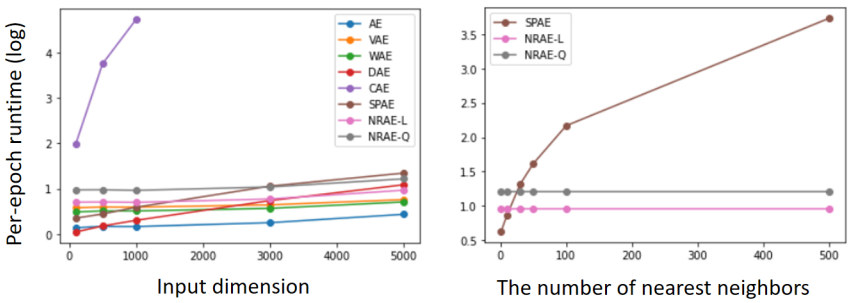


Figure A.1: The per-epoch runtimes as the functions of the input dimension (left) and the number of nearest neighbors (right).

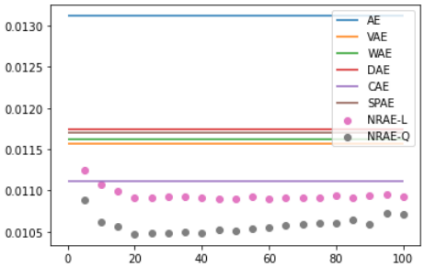


Figure A.2: Comparisons of the test mean reconstruction errors of NRAE and baseline AEs. For NRAE, we report the results as a function of the number of nearest neighbors.

A.2.3 Robustness to the Choice of the Batch Size

In the main script, we set the neighborhood batch size two. We have conducted additional experiments to test the robustness of our approach to the choice of neighborhood batch size (We use MNIST data with a 16-dimensional latent space). For batch sizes (2, 4, 6, 8, 10, 12), the corresponding reconstruction losses of NRAE-L are (0.00953, 0.00952, 0.00950, 0.00952, 0.00955, 0.00945), while those of NRAE-Q are (0.00968, 0.00966, 0.00975, 0.00983, 0.00975, 0.00982), which we think is quite robust.

A.2.4 Extension of the Table 3.3

In Table 3.3 of the main script, we only report the test MSE as a measure of the generalization performance due to the space limitation. To better understand the algorithms, here we report the following additional measures: the Frechet-Inception Distance (FID) scores and the Evidence Lower Bound (ELBO). Also, to see the variances of the algorithms, we repeat the experiments with small datasets five times to compute and report the standard errors of the test MSEs.

FID score and ELBO While the mean reconstruction error is one of the most intuitive methods to measure the difference between data, there are other similarity measures frequently used in the community. The Frechet-Inception Distance (FID) score is a measure of similarity between two datasets of images. It was shown to correlate well with human judgement of visual quality and is most often used to evaluate the quality of generated samples. FID score is calculated by computing the Fréchet distance between two Gaussians fitted to latent feature representations of two sets of images. We measure the FID score with a provided model from `github.com/mseitzer/pytorch-fid`.

As another way to measure the quality of the learned manifold or generalization performance, we convert the trained deterministic models to stochastic models and compare the Evidence Lower Bound (ELBO) (a lower bound of log probability) evaluated with the test data.

Once f_θ and g_ϕ of autoencoder are trained, adopting the idea in [38], we define a latent variable model to estimate $p_{\mathcal{D}}(x)$ as follows:

$$p_{\sigma,\gamma}(x) = \int_z p_\sigma(x|z)p_\gamma(z)dz, \tag{A.2.1}$$

where $p_\gamma(z)$ is a parametric density model such as the normalizing flow models [126, 59, 127, 128], and $p_\sigma(x|z)$ is the stochastic decoder defined as the Gaussian ansatz:

$$p_\sigma(x|z) := \frac{1}{\sqrt{(2\pi)^n \prod_{i=1}^n \sigma_i^2}} \exp(-\frac{1}{2}\sum_{i=1}^n \frac{(x_i - (f_\theta(z)_i))^2}{\sigma_i^2}), \tag{A.2.2}$$

where the noise covariance is the diagonal matrix with $\sigma = (\sigma_1, \ldots, \sigma_n)$. We interpret the deterministic encoder function g_θ as a stochastic encoder $q_\epsilon(z|x) = \mathcal{N}(g_\theta(x), \epsilon^2\mathbb{I})$ with a learnable scalar parameter ϵ, and train $p_{\sigma,\gamma}$ and $q_\epsilon(z|x)$ by maximizing the evidence lower bound (ELBO) as in variational autoencoders training [25] (θ, ϕ are

fixed during training), where the ELBO at data point x is obtained as follows:

$$\mathrm{ELBO}(x) = \mathbb{E}_{q_\epsilon(z|x)}[\log p_\sigma(x|z)] - D_{\mathrm{KL}}(q_\epsilon(z|x)||p_\gamma(z)), \tag{A.2.3}$$

where D_{KL} is the KL-divergence. We use the realnvp model [59] for p_γ where the depth is 8, the lengths of hidden vectors are 32, and resale and permutation are set true.

Test Reconstruction MSEs, FID scores, and ELBOs. As shown in Table A.2, the ELBOs are likely to be high if the MSEs are low because the learned density functions tend to assign high probability densities to the test data whose reconstruction errors are low. The NRAE-L and NRAE-Q mostly show lower MSEs and higher ELBOs than the other baselines. However, for some cases such as the FMNIST, KMNIST, and Omniglot, the CAE or SPAE produce higher ELBOs even though their MSEs are higher than the NRAEs. This is because the ELBOs not only depend on the quality of the learned manifolds but also how easily their encoded latent space distributions can be fit with the normalizing flow models p_γ (i.e., minimizing the KL-divergence term). The MSE and ELBO results in Table A.2 imply that the NRAE-L and NRAE-Q i) mostly learn better manifolds (i.e. low MSEs) than the other baselines yet sometimes ii) produce latent space distributions that are difficult to be learned with p_γ. Although studying how to train the autoencoder in a way that its latent space distribution can be easily learned is an out-of-scope of this thesis, developing a new regularization method that can be added to the NRAE for making its latent space distribution easier to be learned would be an interesting future direction.

On the other hand, the FID scores are not always positively correlated to the MSEs. Since our algorithms are implemented using the Euclidean distance metric for graph construction, the FID scores may not be lower than the others. Nevertheless, for some examples especially small datasets, the NRAE-L and NRAE-Q produce not only lower MSEs but also lower FID scores. For example, the NRAE-L and NRAE-Q show lower MSEs and FID scores for SVHN (S,L), CIFAR10 (S), and CIFAR100 (S), but show lower MSEs yet higher FID scores for CIFAR10 (L), CELEBA (L).

Figure A.3 shows some of the test image reconstruction results of SVHN (S) and CIFAR10 (S). Not only the MSEs and FID scores are lower, but also visual qualities of the reconstructed images by the NARE-L and NRAE-Q are much better than the other baselines. Figure A.4 shows some of the test image reconstruction results of CIFAR10 (L) and CELEBA (L). For CIFAR10 (L), reconstructed results are not significantly different, visually. For CELEBA (L), where the NRAE-L and NRAE-Q show the lowest MSEs but the highest FID scores, reconstructed results of the NRAEs are little

more blurry than the others. Our algorithms seem to overly smooth out the CELEBA data. We believe that this can be alleviated either by decreasing the number of nearest neighbors or decreasing λ. Although we didn't have enough time to empirically prove that NRAEs can better perform on CELEBA (L) in FID scores, in principle, there exist proper number of nearest neighbors and λ with which NRAEs perform at least better or equal to the vanilla AE in FID scores, because the NRAEs have the convergence (to the vanilla AE) property.

It is remarkable that, especially for SVHN, CIFAR10, CIFAR100, CELEBA datasets, the NRAE-L and NRAE-Q trained with small datasets i) largely outperform the other baselines trained with small datasets and ii) show comparable performances compared to the other baselines trained with large datasets. This shows that our algorithm, by leveraging the local geometric information contained in the neighborhood graph, has a significant advantage in generalization when the number of training data is small.

Figure A.3: The test image data reconstruction results where the NRAE-L and NRAE-Q show lower MSEs and FID scores than the other baselines. (left) SVHN (S), (right) CIFAR10 (S).

Standard errors of MSEs (5 times run). In this section, we report the means and standard errors for 5 times run of NRAE and baseline AEs for small datasets. The hyperparameters during 5 times run for each AE are same with the settings whose results are reported in Table 3.3. In table 3.3, we report the means and standard errors of the test reconstruction MSEs. As shown in the table, the standard errors are negligible and the NRAE-L and NRAE-Q show lower MSEs than the other baselines.

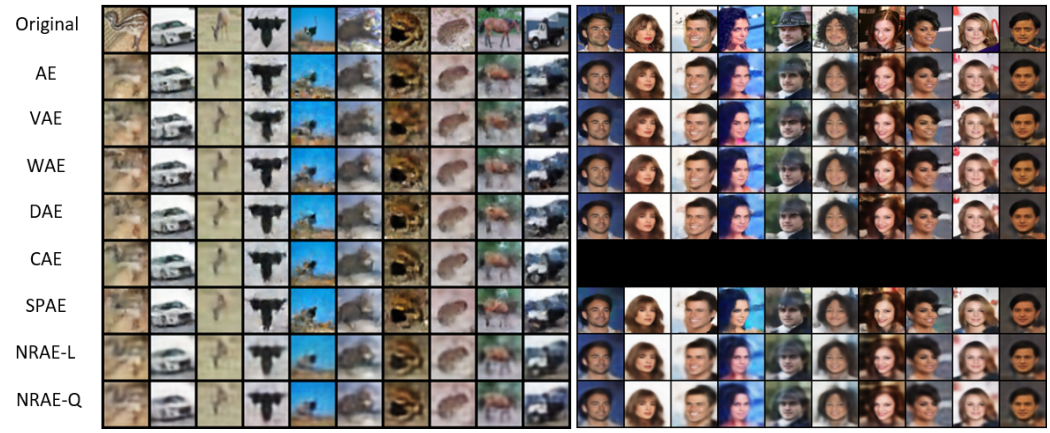


Figure A.4: The test image data reconstruction results where the NRAE-L and NRAE-Q show lower MSEs but higher FID scores than the other baselines. (left) CIFAR (L), (right) CELEBA (L).

A.3 Further Discussion

Relation to local polynomial regression. Smoothing data points by fitting a local polynomial model is a well-known technique in non-parametric regression [129]. Assuming a fixed encoder function g, training the decoder in NRAE shares some similarities to local polynomial regression. For example, given a set of paired data $\mathcal{D}_p := \{(z, x) | z = g(x), x \in \mathcal{D}\}$, the local linear regression problem for estimating f at z is typically formulated as follows:

$$f(z), A^*(z) = \underset{x \in \mathbb{R}^n, A \in \mathbb{R}^{n \times m}}{\arg\min} \sum_{(z', x') \in \mathcal{D}_p} K(z', z) \cdot \|x' - (x + A(z' - z))\|^2, \qquad \text{(A.3.4)}$$

where $K(z', z)$ is a kernel function, $f(z)$ is the estimate of x at z, and $A^*(z) \in \mathbb{R}^{n \times m}$ is the estimated linear coefficient at z. While local polynomial regression is a non-parametric technique that requires solving an optimization for every query point z, NRAE learns a parametric model f_θ for a similar-looking loss function that uses a local polynomial approximation of f_θ.

Convergence to vanilla AE. NRAE is a generalization of a vanilla AE in the following sense: NRAE converges to the vanilla AE – that is, the neighborhood reconstruction loss converges to the point reconstruction loss – if $\mathcal{N}(x) \to \{x\}$ or $K(x', x) \to \delta(x', x)$.

Linear decoder function. As a special case, consider an autoencoder with a linear decoder function such that $f_\theta(z) = \theta_1 z + \theta_0$ for $\theta_1 \in \mathbb{R}^{n \times m}, \theta_0 \in \mathbb{R}^n$. The second-order derivative of f_θ is zero while the first-order derivative $\frac{\partial f_\theta(z)}{\partial z} = \theta_1$; the neighborhood reconstruction loss for a point $x' \in \mathcal{N}(x)$ then becomes

$$\|x' - \tilde{F}_{\theta,\phi}(x'; x)\|^2 := \|x' - \theta_1 g_\phi(x) - \theta_0 - \theta_1(g_\phi(x') - g_\phi(x))\|^2 = \|x' - F_{\theta,\phi}(x')\|^2. \tag{A.3.5}$$

The above implies that NRAE with a linear decoder function is identical to the vanilla AE with the linear decoder.

Table A.2: The test reconstruction MSEs, FID scores (the lower the better) and ELBO (the higher the better). The FID scores are computed on RGB-image datasets only. The best and second-best are colored red and blue, respectively.

Dataset	Metric	Size	AE	VAE	WAE	DAE	CAE	SPAE	NRAE-L	NRAE-Q
MNIST	MSE	S	0.01002	0.01091	0.01009	0.00999	0.00998	0.00989	0.00953	0.00968
		L	0.00688	0.00756	0.00690	0.00684	0.00692	0.00694	0.00649	0.00683
	ELBO	S	202.59	209.31	208.34	201.38	252.31	239.96	371.01	413.16
		L	327.65	223.62	375.6	366.38	458.34	375.62	631.17	658.87
FMNIST	MSE	S	0.01485	0.01652	0.01428	0.01446	0.01319	0.01363	0.01289	0.01277
		L	0.01118	0.01235	0.01106	0.01099	0.01052	0.01065	0.01060	0.01044
	ELBO	S	311.30	285.37	347.14	320.51	494.85	421.59	410.97	430.03
		L	434.51	398.10	443.26	443.44	580.62	533.97	498.14	505.10
KMNIST	MSE	S	0.03267	0.03234	0.03283	0.03280	0.03279	0.03268	0.03071	0.03021
		L	0.02844	0.02963	0.02776	0.02814	0.02762	0.02732	0.02564	0.02602
	ELBO	S	-19.19	22.17	12.98	-23.78	96.98	63.03	42.89	59.58
		L	35.96	43.65	66.34	42.58	174.05	131.77	112.32	120.35
Omniglot	MSE	S	0.03038	0.03627	0.03078	0.03068	0.02714	0.02889	0.02668	0.02631
		L	0.02704	0.03192	0.02728	0.02696	0.02567	0.02644	0.02578	0.02539
	ELBO	S	33.00	-24.62	35.93	30.60	150.62	110.65	96.96	117.34
		L	92.10	20.78	90.65	97.89	189.11	141.95	132.96	148.22
SVHN	MSE	S	0.00320	0.00420	0.00320	0.00369	0.00273	0.00307	0.00202	0.00192
		L	0.00174	0.00204	0.00190	0.00177	0.00178	0.00175	0.00148	0.00147
	ELBO	S	1146.20	987.80	1015.34	1145.97	1304.49	1291.12	3908.37	4050.09
		L	3576.44	3307.51	3100.47	3567.15	4330.74	4130.41	5134.48	4762.67
	FID	S	91.69	124.51	105.00	90.54	60.48	77.38	31.54	28.61
		L	40.44	40.88	40.16	38.34	41.02	40.20	36.95	35.95
CIFAR10	MSE	S	0.01466	0.01620	0.01431	0.01427	0.01208	0.01504	0.00768	0.00691
		L	0.00960	0.01123	0.00863	0.00900	0.00755	0.00898	0.00629	0.00587
	ELBO	S	565.04	269.54	398.32	547.34	908.45	768.79	1963.12	1823.59
		L	520.46	342.21	425.80	631.54	1813.96	930.47	2607.25	2643.33
	FID	S	137.12	157.18	132.79	133.82	108.39	122.20	94.27	85.73
		L	77.43	94.43	71.51	74.91	62.14	66.05	68.74	70.53
CIFAR100	MSE	S	0.01465	0.01713	0.01463	0.01484	0.01369	0.01477	0.00765	0.00717
		L	0.01015	0.01064	0.00951	0.00862	0.00842	0.00912	0.00678	0.00635
	ELBO	S	571.81	257.07	522.22	508.72	812.30	772.32	2004.62	1603.06
		L	625.13	354.69	481.51	468.68	1446.45	917.11	2463.61	2498.37
	FID	S	122.28	139.20	131.91	156.50	127.29	108.86	85.56	80.06
		L	81.35	86.96	78.77	73.95	65.02	66.68	64.84	68.02
CELEBA	MSE	S	0.00780	0.00937	0.00830	0.00782	-	0.00861	0.00608	0.00747
		L	0.00613	0.00646	0.00630	0.00590	-	0.00665	0.00563	0.00565
	ELBO	S	4298.56	4101.55	6043.03	4311.07	-	4513.48	12934.30	11692.26
		L	11224.41	10628.09	11146.31	11634.83	-	11328.64	13456.26	13457.03
	FID	S	60.40	70.13	60.40	59.92	-	60.15	57.51	68.59
		L	43.02	45.18	44.74	42.55	-	43.54	57.70	55.28

Table A.3: The means and standard errors of the test reconstruction MSEs (the lower the better). The metrics are computed with 5 times run except that metrics on the CELEBA data are computed with 3 times run. The best and second-best results are colored red and blue, respectively.

Dataset	Statistic	AE	VAE	WAE	DAE	CAE	SPAE	NRAE-L	NRAE-Q
MNIST	mean	0.010670	0.01094	0.01075	0.01065	0.01037	0.01064	0.00971	0.01013
	ste	±0.00028	±0.00006	±0.00031	±0.00026	±0.00023	±0.00034	±0.00018	±0.00020
FMNIST	mean	0.01435	0.01656	0.01391	0.01403	0.01303	0.01342	0.01281	0.01273
	ste	±0.00015	±0.00003	±0.00013	±0.00014	±0.00008	±0.00009	±0.00003	±0.00003
KMNIST	mean	0.03254	0.03251	0.03306	0.03283	0.03213	0.03255	0.03053	0.02996
	ste	±0.00008	±0.00008	±0.00013	±0.00027	±0.00026	±0.00019	±0.00012	±0.00010
Omniglot	mean	0.03028	0.03146	0.03155	0.03086	0.02891	0.02886	0.02684	0.02631
	ste	±0.00018	±0.00109	±0.00109	±0.00020	±0.00040	±0.00011	±0.00006	±0.00016
SVHN	mean	0.00323	0.00431	0.00319	0.00333	0.00271	0.00310	0.00205	0.00220
	ste	±0.00002	±0.00012	±0.00004	±0.00009	±0.00004	±0.00005	±0.00002	±0.00019
CIFAR10	mean	0.01472	0.01709	0.01445	0.01486	0.01251	0.01465	0.00816	0.00700
	ste	±0.00032	±0.00026	±0.00022	±0.00036	±0.00011	±0.00015	±0.00070	±0.00006
CIFAR100	mean	0.01463	0.017318	0.014548	0.01435	0.01361	0.01455	0.00781	0.00732
	ste	±0.00008	±0.00019	±0.00023	±0.00022	±0.00026	±0.00017	±0.00011	±0.00005
CELEBA	mean	0.00797	0.00914	0.00821	0.00780	-	0.00839	0.00602	0.00727
	ste	±0.00012	±0.00010	±0.00009	±0.00001	-	±0.00009	±0.00003	±0.00008

B

Appendix: Minimum Curvature Manifold Learning

The appendix is organized as follows: (B.1) Related Works, (B.2) Proof of Proposition 4.1, (B.3) On the Extrinsic Curvature, (B.4) Experiment Details, (B.5) Additional Experiment Results, and (B.6) Computational Complexity.

B.1 Related Works: Regularized Autoencoders

The framework of autoencoding together with the recent advances in deep learning techniques used for approximating arbitrary complex functions successfully addresses the manifold learning problem [9]. The core idea is to learn two mappings an encoder $g : \mathbb{R}^D \to \mathbb{R}^m$ and a decoder $f : \mathbb{R}^m \to \mathbb{R}^D$ approximated with deep neural networks so that the composition of them reconstructs the given data points $x_i \in \mathbb{R}^D$, i.e., $f \circ g(x_i) \approx x_i$, for $i = 1, \cdots, N$, and that the data points approximately lie on the image of the decoder, which we refer to as the learned manifold.

Many existing autoencoder regularization methods have focused on the representation learning perspective of autoencoders and studied how to regularize the latent space distributions for purposes like sampling, topology and geometry preserving, clustering, or capturing hierarchical structure [32, 25, 34, 26, 27, 29, 30, 31, 19, 20, 21, 22]; since the latent space distributions are entirely determined by the encoders, they mostly focus on regularizing the encoders but not decoders.

As discovered in [13], to learn the accurate manifold in the presence of data noise or given a small number of training data, regularization of the decoder is indeed more important, because it is the decoder that has information about how the manifold lies in the data space. Based on the intuition that a local approximation of the decoder contains local geometric information on the decoded manifold (i.e. learned manifold), e.g., a local linear approximation of f spans the tangent space, a priori constructed

neighborhood graph is employed to regularize the local approximation of the decoder and hence the decoded manifold. This has shown improved manifold learning accuracy for both noisy and small training dataset cases, however obviously, the performance largely depends on the quality of the graph as in many other graph-based methods.

There are graph-free autoencoder regularization methods that regularize not only an encoder but also the decoder. Denoising autoencoder [18] is trained to reconstruct a corrupted input to its clean version with the following loss

$$\sum_{i=1}^{N} \|x_i - f(g(x_i + \epsilon))\|^2, \tag{B.1.1}$$

for some noise variable ϵ. As a limit case in [41], the Jacobian of the reconstruction function is minimized where the loss is defined as follows:

$$\sum_{i=1}^{N} \|x_i - f(g(x_i))\|^2 + \alpha \ \|\frac{\partial f \circ g}{\partial x}(x_i)\|_F^2, \tag{B.1.2}$$

where α is the regularization coefficient and $\|\cdot\|_F$ denotes the Frobenius norm. These by construction attempt to learn manifolds robust to noise, but we note that (i) they are designed to be robust to noise during inference after being trained with clean data, but if training data points themselves are noisy, the robust manifold learning performance decreases and (ii) their regularization effects are limited to where data points are available.

Since regularizing the decoder that explicitly parameterizes the manifold is important, one may consider minimizing the norm of decoder's Jacobian as

$$\sum_{i=1}^{N} \|x_i - f(g(x_i))\|^2 + \alpha \ \|\frac{\partial f}{\partial z}(g(x_i))\|_F^2 \tag{B.1.3}$$

with the regularization coefficient α or the norm of decoder's Hessian $\sum_{i,j} \|\frac{\partial^2 f}{\partial z_i \partial z_j}(g(x_i))\|^2$. However, these norms do not capture geometric quantities of the learned manifold because they are not coordinate-invariant or reparmetrization-invariant, and thus they do not produce any meaningful regularization effects.

For example, consider a coordinate transformation $z' = h(z)$ which converts the encoder as $g \mapsto g' = h \circ g$ and decoder as $f \mapsto f' = f \circ h^{-1}$. The reconstruction loss is invariant since $f \circ g = f' \circ g'$, and hence the learned manifold is invariant, but the

regularization term, the norm of decoder's Jacobian, is different:

$$\|\frac{\partial f'}{\partial z'}(g'(x_i))\|_F^2 = \|\frac{\partial f}{\partial z}(g(x_i))\frac{\partial h^{-1}}{\partial z'}(g'(x_i))\|_F^2 \neq \|\frac{\partial f}{\partial z}(g(x_i))\|_F^2. \tag{B.1.4}$$

This implies that we can minimize the norm of decoder's Jacobian just by increasing the norm of Jacobian of h^{-1} without actually changing the learned manifold. A similar argument holds for the Hessian norm.

Recent works [33, 14] have suggested decoder regularization methods for learning isometric representations that preserve geometry of the data space. A common goal is to learn a decoder $f : \mathbb{R}^m \to \mathbb{R}^D$ that satisfies

$$\frac{\partial f}{\partial z}(z)^T \frac{\partial f}{\partial z}(z) = cI \text{ for all } z \in \nu(\mathbb{R}^m) \tag{B.1.5}$$

for some positive scalar c, where I is the $m \times m$ identity matrix and $\nu(\mathbb{R}^m)$ is the support of the latent space data distribution. Such mappings are formally defined as scaled isometries in [14], which are geometry-preserving mappings in the sense that latent space straight lines are mapped to the geodesic curves in the learned manifold.

Regularizing the decoder to be a scaled isometry, beyond finding geometry-preserving representations, has an implicit manifold regularization effect. According to Gauss's Theorema Egregium which states that "The Gaussian curvature of a surface is invariant under local isometry", for scaled isometries f to exist, the Gaussian curvature of the learned manifold should be the same as that of the Euclidean space (i.e., zero). In other words, it has an *implicit intrinsic curvature minimization* effect, which is different from the method proposed in this paper that explicitly minimizes the extrinsic curvature.

B.2 Proof of Proposition 4.1

Proof. Let's denote by

$$c(\theta, \phi) = \sum_{i,j} (J_\theta^T J_\theta)_{ij}^{-1} \mathsf{Tr}(\frac{\partial E(J_\theta)}{\partial z_i} \frac{\partial E(J_\theta)}{\partial z_j}).$$

Given a coordinate transformation $z' = h(z)$ that maps $(g_\phi, f_\theta) \mapsto (g_{\phi'}, f_{\theta'}) = (h \circ g_\phi, f_\theta \circ h^{-1})$, the following transformation rules hold: $J_\theta \mapsto J_{\theta'} = J_\theta \cdot \frac{\partial h^{-1}}{\partial z'}$ and $\frac{\partial I}{\partial z} \mapsto \frac{\partial I}{\partial z'} = \frac{\partial I}{\partial z}\frac{\partial h^{-1}}{\partial z'}$ for some scalar-valued function $I(z)$. We note that, since $E(J) = E(JA)$ for some arbitrary invertible matrix A, the embedding is invariant, i.e., $E(J_{\theta'}) =$

$E(J_\theta)$. Let $I_{\alpha\beta}$ denote the (α,β)-component of $E(J_\theta)$, then, by using $\mathsf{Tr}(AB) = \sum_{\alpha,\beta} A_{\alpha\beta}B_{\beta\alpha}$ and denoting $\frac{\partial h^{-1}}{\partial z'}$ by H,

$$\begin{aligned} c(\theta,\phi) \mapsto c(\theta',\phi') &= \sum_{i,j}(J_{\theta'}^T J_{\theta'})^{-1}_{ij} \sum_{\alpha,\beta} \frac{\partial I_{\alpha\beta}}{\partial z'_i}\frac{\partial I_{\beta\alpha}}{\partial z'_j} \\ &= \sum_{\alpha,\beta} \frac{\partial I_{\alpha\beta}}{\partial z'}(J_{\theta'}^T J_{\theta'})^{-1}\big(\frac{\partial I_{\beta\alpha}}{\partial z'}\big)^T \\ &= \sum_{\alpha,\beta} \frac{\partial I_{\alpha\beta}}{\partial z} H(H^T J_\theta^T J_\theta H)^{-1} H^T \big(\frac{\partial I_{\beta\alpha}}{\partial z}\big)^T \\ &= \sum_{\alpha,\beta} \frac{\partial I_{\alpha\beta}}{\partial z}(J_\theta^T J_\theta)^{-1}\big(\frac{\partial I_{\beta\alpha}}{\partial z}\big)^T \\ &= \sum_{i,j}(J_\theta^T J_\theta)^{-1}_{ij} \sum_{\alpha,\beta} \frac{\partial I_{\alpha\beta}}{\partial z_i}\frac{\partial I_{\beta\alpha}}{\partial z_j} = c(\theta,\phi). \end{aligned} \tag{B.2.6}$$

□

B.3 On the Extrinsic Curvature Measure

In this section, we we will derive the expression of our extrinsic curvature measure

$$\sum_{i,j}(J^T J)^{-1}_{ij}\mathsf{Tr}\big(\frac{\partial J(J^T J)J^T}{\partial z^i}\frac{\partial J(J^T J)J^T}{\partial z^j}\big)$$

for a one-dimensional manifold, i.e., a curve, embedded in $\mathbb{R}^D$. Let $x : \mathbb{R} \to \mathbb{R}^D$ be a smooth curve and assume that it is parameterized by arc-length, i.e., $\|\frac{\partial x}{\partial z}\| = J^T J = 1$.

Then, the curvature becomes

$$\begin{aligned}
\mathsf{Tr}((\frac{\partial J J^T}{\partial z})^2) &= \mathsf{Tr}((\frac{\partial}{\partial z}(\frac{\partial x}{\partial z}\frac{\partial x}{\partial z}^T))^2) \\
&= \mathsf{Tr}((\frac{\partial^2 x}{\partial z^2}\frac{\partial x}{\partial z}^T + \frac{\partial x}{\partial z}\frac{\partial^2 x}{\partial z^2}^T)^2) \\
&= \mathsf{Tr}(\frac{\partial^2 x}{\partial z^2}\frac{\partial x}{\partial z}^T\frac{\partial^2 x}{\partial z^2}\frac{\partial x}{\partial z}^T + 2\frac{\partial^2 x}{\partial z^2}\frac{\partial x}{\partial z}^T\frac{\partial x}{\partial z}\frac{\partial^2 x}{\partial z^2}^T + \frac{\partial x}{\partial z}\frac{\partial^2 x}{\partial z^2}^T\frac{\partial x}{\partial z}\frac{\partial^2 x}{\partial z^2}^T) \\
&= \frac{\partial x}{\partial z}^T\frac{\partial^2 x}{\partial z^2}\frac{\partial x}{\partial z}^T\frac{\partial^2 x}{\partial z^2} + 2\frac{\partial x}{\partial z}^T\frac{\partial x}{\partial z}\frac{\partial^2 x}{\partial z^2}^T\frac{\partial^2 x}{\partial z^2} + \frac{\partial^2 x}{\partial z^2}^T\frac{\partial x}{\partial z}\frac{\partial^2 x}{\partial z^2}^T\frac{\partial x}{\partial z} \\
&= 2(\frac{\partial x}{\partial z}^T\frac{\partial^2 x}{\partial z^2})^2 + 2\frac{\partial^2 x}{\partial z^2}^T\frac{\partial^2 x}{\partial z^2}. \qquad \text{(B.3.7)}
\end{aligned}$$

Since $\frac{\partial}{\partial z}\|\frac{\partial x}{\partial z}\| = 0$ implies that $\frac{\partial x}{\partial z}^T\frac{\partial^2 x}{\partial z^2} = 0$, our curvature measure for an arc-length parameterized curve $x(z)$ is simplified to $2\|\frac{\partial^2 x}{\partial z^2}\|$ that is twice the norm of second derivative. This is equivalent to the classical definition of the curvature of a curve.

B.4 Experiment Details

B.4.1 Grayscale Image Data

The image size is 28 $\times$ 28 and the pixel values are normalized between 0 and 1. The encoder and decoder are two-layer fully connected neural networks with the ELU activation functions and 512 nodes for each layer. The output layer is linear for the encoder and sigmoid for the decoder. For clean dataset cases, we use the following early stopping criteria in training: we stop the training if the mean reconstruction error for the validation dataset increases 10 times in a row; then we use the best model (i.e. the lowest validation errors) for evaluation. For noisy dataset cases, assuming that we don't have an access to the clean dataset during training, we do not use the early stopping and trained the model for a sufficiently big number of epochs for convergence (the number of epochs is 1000). The number of test data is 60000. For evaluation, we use clean test data for noisy training dataset cases as well. The batch size is 100 and the learning rate is 0.001.

B.4.2 SVHN & CIFAR10 Image Data

The image size is 32 $\times$ 32 and the pixel values are normalized between 0 and 1. For noisy training dataset experiments, we add noises as follows: (i) for Gaussian noise,

the standard deviation is 0.1, (ii) for Shot noise, we multiply 0.15 to noise variables sampled from the Poisson distributions where λ are image pixel values, and (iii) for Impulse noise, with 5% probability we randomly add 1 to each pixel. The encoder and decoder are convolutional and transposed convolutional neural networks with the ReLU activation functions, where, denoting a convolution layer of input channel size c_i, output channel size c_o, kernel size k, stride s, and padding p by Conv2d(c_i, c_o, k, s, p) and transposed convolution layer by ConvTrans2d(c_i, c_o, k, s, p), the following sequence of layers Conv2d(3, 128, 4, 2)-Conv2d(128, 256, 4, 2)-Conv2d(256, 512, 4, 2)-Conv2d(512, 1024, 2, 2)-Conv2d(1024, 64, 1) is used for encoder and ConvTrans2d(64, 1024, 8)-ConvTrans2d(1024, 512, 4, 2, 1)-ConvTrans2d(512, 256, 4, 2, 1)-ConvTrans2d(512, 3, 1) for decoder. The output layer is linear for the encoder and sigmoid for the decoder. For clean dataset cases, we use the following early stopping criteria in training: we stop the training if the mean reconstruction error for the validation dataset increases 10 times in a row; then we use the best model (i.e. the lowest validation errors) for evaluation. For noisy dataset cases, assuming that we don't have an access to the clean dataset during training, we do not use the early stopping and trained the model for a sufficiently big number of epochs for convergence (the number of epochs is 100). The number of test data is 63257 for SVHN and 10000 for CIFAR10. For evaluation, we use clean test data for noisy training dataset cases as well. The batch size is 8 and the learning rate is 0.0001.

B.4.3 Human Skeleton Pose Data

From the NTU RGB+D dataset, a set of human pose skeleton data that consists of 25 key points is extracted and pre-processed to be aligned. Specifically, 10000 poses are extracted from each action class (a total of 60 action classes is used), and they are rotated and translated so that the 1-2 key points direction becomes z-axis and 1-13 key points direction becomes the y-axis and the key point number 2 becomes the origin. The encoder and decoder are two-layer fully connected neural networks with the ELU activation functions and 512 nodes for each layer. The output layers are linear for both the encoder and decoder. For clean dataset cases, we use the following early stopping criteria in training: we stop the training if the mean reconstruction error for the validation dataset increases 10 times in a row; then we use the best model (i.e. the lowest validation errors) for evaluation. For noisy dataset cases, assuming that we don't have an access to the clean dataset during training, we do not use the early stopping and trained the model for a sufficiently big number of epochs for convergence (the number of epochs is 5000). The number of test data is 9000. For evaluation, we use clean test data for noisy training dataset cases as well. The batch size is 100 and the learning rate is 0.0001.

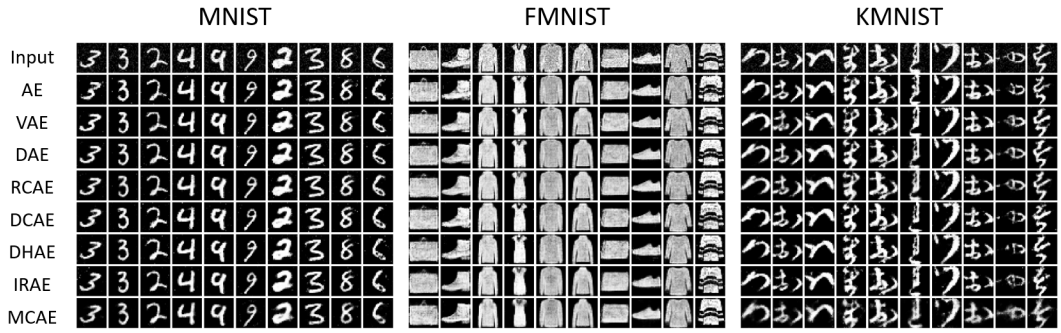


Figure B.1: De-noising examples of grayscale image data (noise level 0.1).

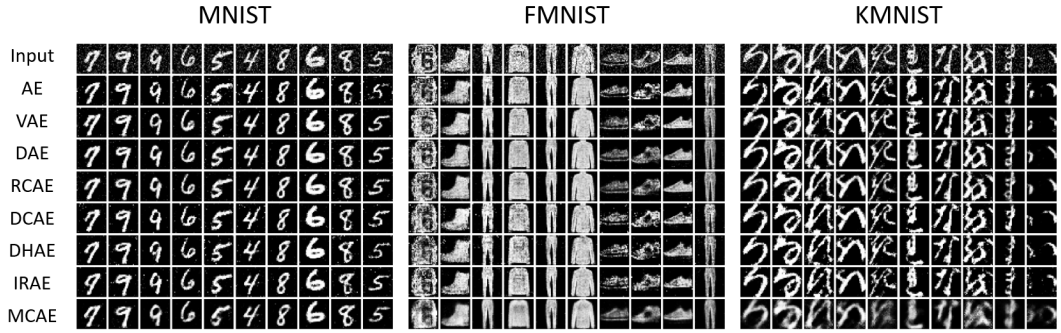


Figure B.2: De-noising examples of grayscale image data (noise level 0.2).

B.5 Additional Experiment Results

B.5.1 More Qualitative Results

Figure B.1, B.2, B.3, B.4, B.5, B.6 show additional de-noising results for image data and human skeleton pose data.

B.6 Computational Complexity

In this section, we provide actual computation time of the curvature measure in (4.3.7) and backpropagation time with two-layer fully connected neural networks used for gray-scaled image data and convolutional and transposed convolutional neural networks used for the SVHN and CIFAR10 image data. Throughout this study, the NVIDIA GeForce RTX 3090 is used.

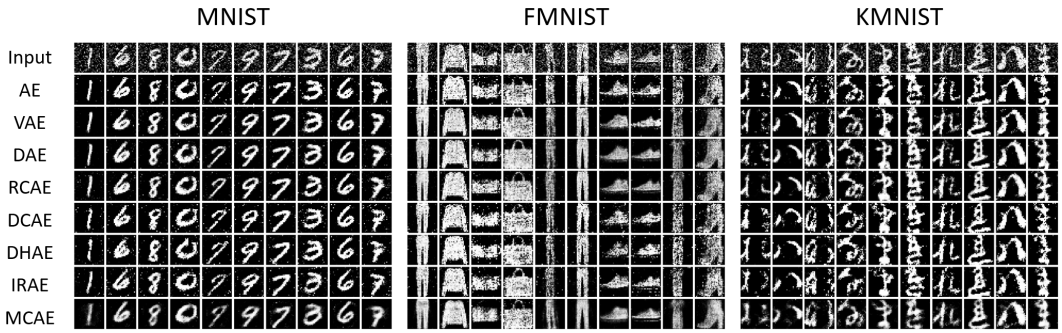


Figure B.3: De-noising examples of grayscale image data (noise level 0.3).

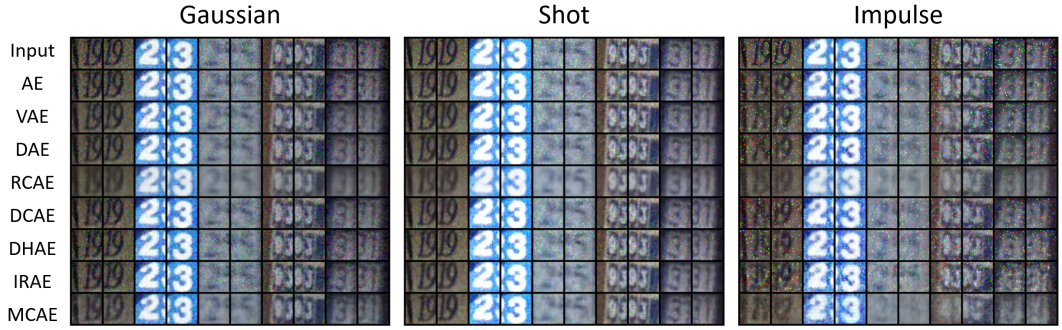


Figure B.4: De-noising examples of SVHN data.

Figure B.5: De-noising examples of CIFAR10 data.

Table B.1 shows the per-batch computation time comparisons between reconstruction loss term and curvature term in (4.3.7). Although the curvature computation has

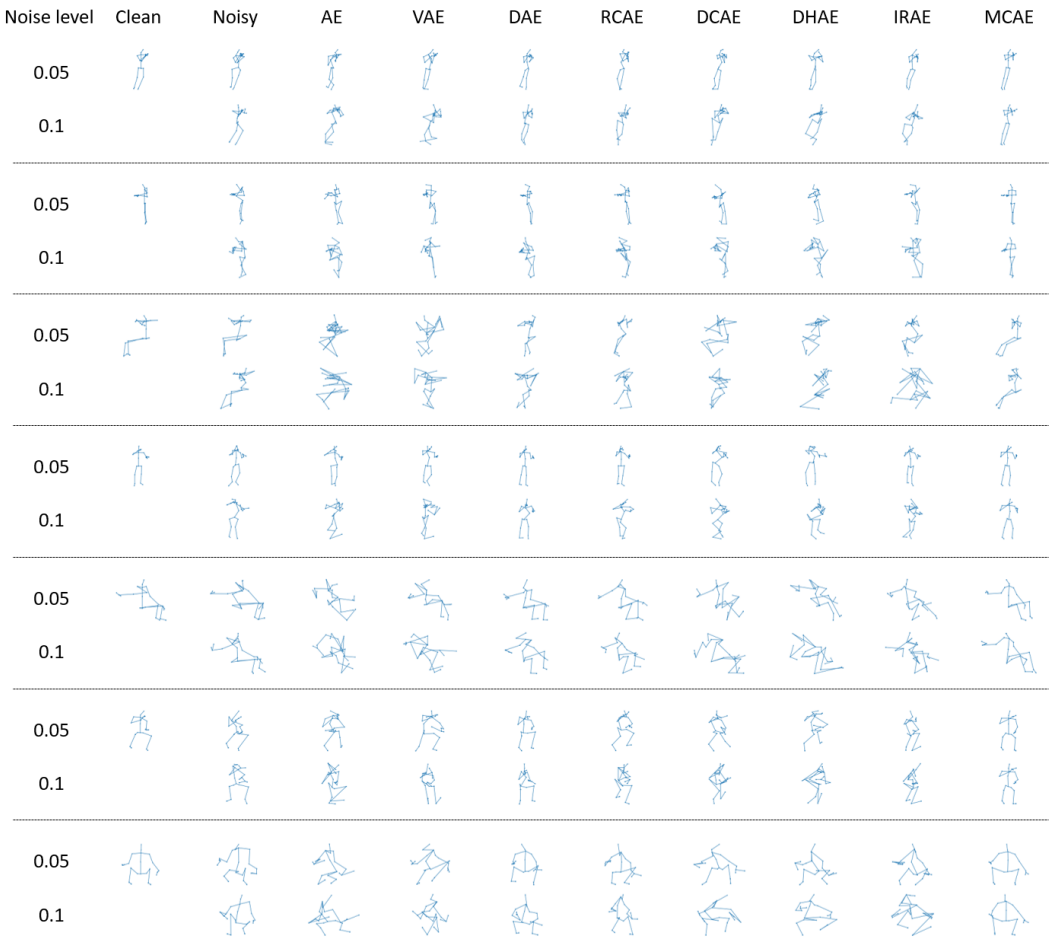


Figure B.6: De-noising examples of human pose data.

become feasible through the stochastic trace estimation, compared to the original reconstruction loss term, it still takes much longer time. Especially, looking at the forward computation time for the Conv net case, the curvature computation is almost 100 to 150 times slower than the reconstruction term computation.

To see which part in the below curvature measure

$$\mathcal{C}(\theta, \phi) = \mathbb{E}_{z\sim\hat{p}_\phi(z), v\sim\mathcal{N}(0,I_m), w\sim\mathcal{N}(0,I_D)}[v^T \frac{\partial(w^T E(J_\theta))}{\partial z} \frac{\partial(E(J_\theta)w)}{\partial z} G_\theta^{-1} v]$$

Table B.1: Per-batch computation time comparisons. For the FC net with $1 \times 28 \times 28$ image, the latent space dimension is 16 and the batch size is 100, and for the Conv net with $3 \times 32 \times 32$ image, the latent space dimension is 64 and the batch size is 8 (for GPU memory limitation).

	FC net with $1 \times 28 \times 28$ image		Conv net with $3 \times 32 \times 32$ image	
	Forward Computation	Back-Propagation	Forward Computation	Back-Propagation
Reconstruction	0.00037 s	0.00045 s	0.00220 s	0.00114 s
Curvature	0.00967 s	0.00447 s	0.30147 s	0.06051 s

requires a major computational cost, we compare the computation times of the following operations: (i) the Riemannian metric $G_\theta = J_{f_\theta}^T J_{f_\theta}$, (ii) the inverse of G_θ, (iii) the Jacobian-vector product for $\frac{\partial(E(J_\theta)w)}{\partial z}v$, and (iv) the Jacobian-vector product for $\frac{\partial(E(J_\theta)w)}{\partial z}(G_\theta^{-1}v)$.

Table B.2 shows the per-batch computation times of the intermediate operations in curvature measure (4.3.7) for the Conv net case. As can be seen, the inverse computation takes up most of the total computation time. To reduce the computation time of the matrix inverse, one can consider an approximate inverse computation method. For example, given G_θ, let us define a function $f : \mathbb{R}^{m\times m} \to \mathbb{R}^{m\times m}$ such that

$$f(X) = X^{-1} - G_\theta. \tag{B.6.8}$$

To find the root of f, we can use the standard Newton-Raphson method:

$$X_{n+1} = 2X_n - X_n G_\theta X_n, \tag{B.6.9}$$

which is known as the Newton-Schulz iteration method for the matrix inversion. We can get an approximation of G_θ^{-1} by iteratively applying the above, where it gets closer to the true inverse as we increase the number of iteration.

Table B.3 shows the per-batch computation times and percent errors of the approximate matrix inverse G_θ^{-1} in curvature measure (4.3.7) as the number of iteration increases with the Conv net case. The percent error is computed as $100 * \|G_{\text{true}}^{-1} - G_{\text{est}}^{-1}\|_F / \|G_{\text{true}}^{-1}\|_F$. When the number of iterations is set to be 100, the percent error is only 0.01 % while significantly reducing the computation time as 0.28353 s $\to$ 0.005669 s. It is highly recommended to use the approximate matrix inverse when the latent space dimension is high.

Table B.2: Per-batch computation times of the intermediate operations in curvature measure (4.3.7) with the Conv net with $3 \times 32 \times 32$ image and 64-dimensional latent space.

$J_{f_\theta}^T J_{f_\theta}$	G_θ^{-1}	$\frac{\partial(E(J_\theta)w)}{\partial z} v$	$\frac{\partial(E(J_\theta)w)}{\partial z}(G_\theta^{-1} v)$
0.00192 s	0.28353 s	0.00682 s	0.00647 s

Table B.3: Per-batch computation times and percent errors of the approximate matrix inverse G_θ^{-1} in curvature measure (4.3.7) as the number of iteration increases with the Conv net with $3 \times 32 \times 32$ image and 64-dimensional latent space.

number of iterations	ground truth	1	5	10	100	1000
time	0.28353 s	0.000256 s	0.000507 s	0.000807 s	0.005669 s	0.053519 s
percent error	0 %	99.83 %	97.69 %	71.35 %	0.0109 %	0.0095 %

C

Appendix: Regularized Autoencoders for Isometric Representation Learning

C.1 Proof of Propositions

Proof of Proposition 5.3.1. For sufficiency ($\Rightarrow$) of the second condition, suppose $\mathcal{F}(f) = 0$. Then, denoting by $c = \int_{\mathcal{M}} S(\lambda_1(z), ..., \lambda_m(z))\, d\nu(z)$,

$$\sum_{i=1}^{m} h(\frac{\lambda_i(z)}{c}) = 0 \quad \forall z \in \mathrm{Supp}(\nu).$$

Since $h(\lambda) \geq 0$ for $\forall \lambda \in \mathbb{R}$, $h(\lambda_i(z)/c) = 0$ for $\forall i, \forall z \in \mathrm{Supp}(\nu)$. Since $h'(\lambda) = 0$ iff $\lambda = 1$, $\lambda_i(z) = c$ for $\forall i, \forall z \in \mathrm{Supp}(\nu)$.

For necessity ($\Leftarrow$) of the second condition, suppose $\lambda_i(z) = c$ for $\forall i, \forall z \in \mathrm{Supp}(\nu)$, then

$$\begin{aligned}
\frac{\lambda_i(z)}{\int_{\mathcal{M}} S(\lambda_1(z), ..., \lambda_m(z))\, d\nu(z)} &= \frac{c}{\int_{\mathcal{M}} S(c, ..., c)\, d\nu(z)} \\
&= \frac{c}{c \int_{\mathcal{M}} S(1, ..., 1)\, d\nu(z)} \\
&= \frac{1}{\frac{1}{\nu(\mathcal{M})} \int_{\mathcal{M}} d\nu(z)} \\
&= 1 \text{ for } \forall z \in \mathrm{Supp}(\nu).
\end{aligned}$$

Then consequently,

$$\mathcal{F}(f) = \int_{\mathcal{M}} \sum_{i=1}^{m} h(\frac{\lambda_i(z)}{\int_{\mathcal{M}} S(\lambda_1(z), ..., \lambda_m(z))\, d\nu(z)})\, d\nu(z) = \int_{\mathcal{M}} \sum_{i=1}^{m} h(1)\, d\nu(z) = 0.$$

For the third condition, suppose $J_f^T H J_f = c J_g^T H J_g$ in $\mathrm{Supp}(\nu)$ for some $c > 0$. Denote the eigenvalues of $J_f^T H J_f G^{-1}$ and $J_g^T H J_g G^{-1}$ by $\lambda_i[f]$ and $\lambda_i[g]$ where $\lambda_1[f] \geq ... \geq \lambda_m[f]$ and $\lambda_1[g] \geq ... \geq \lambda_m[g]$, respectively. Then, obviously, $\lambda_i[f] = c\lambda_i[g]$ for $\forall i$. Then consequently,

$$\begin{aligned}
\mathcal{F}(f) &= \int_{\mathcal{M}} \sum_{i=1}^{m} h(\frac{\lambda_i[f](z)}{\int_{\mathcal{M}} S(\lambda_1[f](z), ..., \lambda_m[f](z))\, d\nu(z)})\, d\nu(z) \\
&= \int_{\mathcal{M}} \sum_{i=1}^{m} h(\frac{c\lambda_i[g](z)}{\int_{\mathcal{M}} c \cdot S(\lambda_1[g](z), ..., \lambda_m[g](z))\, d\nu(z)})\, d\nu(z) \\
&= \int_{\mathcal{M}} \sum_{i=1}^{m} h(\frac{\lambda_i[g](z)}{\int_{\mathcal{M}} S(\lambda_1[g](z), ..., \lambda_m[g](z))\, d\nu(z)})\, d\nu(z) = \mathcal{F}(g).
\end{aligned}$$

□

Proof of Proposition 5.4.1. Note that $\mathrm{Tr}(H_\theta(z)) = \sum_i \lambda_i(z)$ and $\mathrm{Tr}(H_\theta^2(z)) = \sum_i \lambda_i^2(z)$, and denote by $E := \mathbb{E}_{z\sim P_\phi}[\mathrm{Tr}(H_\theta(z))]$. Then

$$\begin{aligned}
\mathbb{E}_{z\sim P_\phi}[\sum_{i=1}^{m}(\frac{\lambda_i(z)}{\mathbb{E}_{z\sim P_\phi}[\sum_i \lambda_i(z)/m]} - 1)^2] &= \mathbb{E}_{z\sim P_\phi}[\sum_{i=1}^{m}(\frac{m\lambda_i(z)}{E} - 1)^2] \\
&= \mathbb{E}_{z\sim P_\phi}[\sum_{i=1}^{m}(\frac{m^2\lambda_i^2(z)}{E^2} - \frac{2m\lambda_i(z)}{E} + 1)] \\
&= \mathbb{E}_{z\sim P_\phi}[\sum_{i=1}^{m}\frac{m^2\lambda_i^2(z)}{E^2}] - m \\
&= m^2\frac{\mathbb{E}_{z\sim P_\phi}[\mathrm{Tr}(H_\theta^2(z))]}{\mathbb{E}_{z\sim P_\phi}[\mathrm{Tr}(H_\theta(z))]^2} - m.
\end{aligned}$$

□

C.2 Pseudocode

In this section, we provide pytorch style pseudocode for our Isometric Regularization and Flattening Module.

C.2.1 Isometric Regularization of Autoencoders

In the main manuscript, we mainly study the isometric regularization effect on Variational Autoencoder (VAE). VAE is just one possible choice; our method is straightforwardly applicable to other types of autoencoders. We consider a deterministic encoder and decoder in the following pseudocode.

```
'''We assume
x: input data w/ size (batch_size, data_space_dimension)
z: latent value w/ size (batch_size, latent_space_dimension)
encoder: torch.nn.Module (e.g., z = encoder(x))
decoder: torch.nn.Module (e.g., x = decoder(z))
eta: mixup parameter
'''
def relaxed_distortion_measure(decoder, z):
    bs, z_dim = z.size()

    v = torch.randn(bs, z_dim)
    Jv = torch.autograd.functional.jvp(decoder, z, v=v)
    TrG = torch.sum(Jv**2, dim=1).mean()
    JTJv = torch.autograd.functional.vjp(decoder, z, v=JV)
    TrG2 = torch.sum(JTJv**2, dim=1).mean()
    return TrG2/(TrG**2)

def isometric_regularization_term(x, encoder, decoder, eta):
    z = encoder(x)
    bs, z_dim = z.size()

    # sample z_augmented from P_Z
    z_permuted = z[torch.randperm(bs)]
    alpha_samples = (torch.rand(bs, 1) * (1 + 2*eta) - eta)
    z_augmented = alpha_samples*z + (1 - alpha_samples)*z_permuted

    # compute relaxed distortion measure
    return relaxed_distortion_measure(decoder, z_augmented)
```

C.2.2 Flattening Module

```
'''In addition to the above, we assume
z_: new latent value w/ size (batch_size, latent_space_dimension)
```

```
flattening_module: torch.nn.Module (e.g., z_ = flattening_module(z)), invertible
'''

def flattening_loss(x, encoder, decoder, flattening_module, eta):
    z_ = flattening_module(encoder(x))
    bs, z_dim = z_.size()

    z_permuted = z_[torch.randperm(bs)]
    alpha_samples = (torch.rand(bs, 1) * (1 + 2*eta) - eta)
    z_augmented = alpha_samples*z_ + (1 - alpha_samples)*z_permuted

    def func(z_):
        z = inverse_of_flattening_module(z_)
        return decoder(z)

    return relaxed_distortion_measure(func, z_augmented)
```

C.3 Comparison to Other Regularization Approaches that use the Jacobian.

There have been several works on regularized autoencoders that use the Jacobian of the encoder or decoder. In the following discussion, let $g(x)$ be an encoder and $f(z)$ be a decoder, and denote the Jacobian of $g(x)$ by $J_g(x)$ and the Jacobian of $f(z)$ by $J_f(z)$. Let $\|\cdot\|_F$ denote the Frobenius norm. The latent space dimension is denoted by m. P_D is the data distribution and P_Z is the (augmented) encoded data distribution.

The Contractive Autoencoder (CAE) attempts to enhance robustness of representation by penalizing the Jacobian norm of the encoder function [32]. Mathematically, the regularization term is

$$\mathbb{E}_{x\sim P_D}[\|J_g(x)\|_F^2].$$

The Contractive Autoencoder with Hessian regularization (CAE+H) introduced not long after in [130] has regularization term

$$\mathbb{E}_{x\sim P_D}[\|J_g\|_F^2 + \gamma\|J_g(x) - J_g(x+\epsilon)\|_F^2],$$

where ϵ is a small value and γ controls the balance between the two terms. The second order regularization using the Hessian penalizes curvature, and thus favors smooth manifolds. These approaches mainly aim to learn robust representations by finding smooth encoders g that have small Jacobian and Hessian norms.

More recently, a regularization method that finds a geometry preserving mapping

has been introduced [33], called the Flat Manifold Variational Autoencoder (FMVAE). This is the closest work to our thesis, and similarly aims to find a scaled isometry. However, the regularization term introduced in FMVAE has several limitations compared to ours.

We first note that geometric objects that we want to preserve, such as length, angle, and volume, are all coordinate invariant concepts, which require a coordinate invariant formulation from the start. If one uses a coordinate-variant measure, then there is no guarantee that it will work effectively for different choices of coordinates. Also, there is no a priori way of knowing which coordinates are best.

In this regard, we first show that the regularization term defined in FMVAE is not coordinate-invariant. Let $\mathcal{M}$ be a Riemannian manifold of dimension m with local coordinates $z \in \mathbb{R}^m$ and Riemannian metric $G(z) \in \mathbb{R}^{m\times m}$, and $\mathcal{N}$ be a Riemannian manifold of dimension n with local coordinates $x \in \mathbb{R}^n$ and Riemannian metric $H(x) \in \mathbb{R}^{n\times n}$. Let $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ be a smooth mapping, represented in local coordinates by the italic symbol $f : \mathbb{R}^m \to \mathbb{R}^n$. Let J_f be the Jacobian of f and P_Z be the probability distribution expressed in $\mathbb{R}^m$. Let ν be a positive measure on $\mathcal{M}$.

The regularization term defined in FMVAE is then

$$\|J_f^T(z)H(f(z))J_f(z) - cG(z)\|_F^2,$$

where $\|\cdot\|_F$ is the Frobenius norm and $c = \int_{\mathcal{M}} \frac{1}{m}\mathrm{Tr}(J_f^T(z)H(f(z))J_f(z))d\nu$. For simplicity, consider a pair of linear coordinate transformations on the input manifold $z' = Az$ and output manifold $x' = Bx$. Then the function f is transformed to $f'(z') := Bf(A^{-1}z')$, $G(z)$ is transformed to $G'(z') := A^{-T}G(z)A^{-1}$ and $H(x)$ is transformed to $H'(x') := B^{-T}H(x)B^{-1}$. Then, after some calculations, the above regularization term is transformed to

$$\left\|A^{-T}\left(J_f^T(z)H(f(z))J_f(z) - c'G(z)\right)A^{-1}\right\|_F^2,$$

where $c' = \int_{\mathcal{M}} \frac{1}{m}\mathrm{Tr}(A^{-T}J_f^T(z)H(f(z))J_f(z)A^{-1})d\nu$. Recall that the Frobenius norm $\|A\|_F^2 = \frac{1}{2}\mathrm{Tr}(A^TA)$, the A^{-T} and A^{-1} multiplied at sides are not canceled; we can see that this is not coordinate-invariant.

In addition to this, perhaps a more direct reason that the FMVAE does not perform as well as our method can be explained as follows. We remark that one of the desired properties of the scaled isometry measure is the third condition in Section 3.3: given two mappings f and f' from $\mathcal{M}$ to $\mathcal{N}$, if $J_f^T(z)H(f(z))J_f(z) = cJ_{f'}^T(z)H(f'(z))J_{f'}(z)$ for some $c > 0$ for all z, then the scaled isometry measure should be the same (that is, the metric should not distinguish between f and f'). This makes the measure more natural, in the sense that it does not a priori favor a

particular scale for the pullback metric; if the pullback metrics are equivalent up to some scale, then these should be treated the same.

The regularization term introduced in FMVAE can also be viewed as a scaled isometry measure. However, it fails to satisfy the all-important third property. Even though the pullback metrics of f and f' are equivalent up to some scalar multiplication, i.e., $J_f^T(z)H(f(z))J_f(z) = cJ_{f'}^T(z)H(f'(z))J_{f'}(z)$ for some $c > 0$ for all z, the one with the "smaller" Jacobian (i.e., smaller norm) achieves a lower value of the regularization term. As a result, the FMVAE favors a mapping with a smaller Jacobian, which can be detrimental to learning accurate data manifolds.

On the other hand, our relaxed distortion measures are defined in a coordinate-invariant way. Further, as we have shown in our main manuscript, the measure does not favor a particular scale of the pullback metric. We believe, because of these characteristics, IRVAE can outperform FMVAE.

C.4 Experimental Details

C.4.1 Section 5.5.1.1

Dataset: We use MNIST dataset. The training, validation, and test data are 50,000, 10,000, and 10,000, respectively.

Network Architecture: For both FMVAE, IRVAE, we use fully-connected neural networks that have four hidden layers with ReLU activation functions (256 nodes for each layer are used) for encoder and decoder. The output activation functions are linear and sigmoid for encoder and decoder, respectively. For FM, we use the RealNVP model of depth 8 and length 512.

Other Details: For FMVAE and IRVAE, the batch size is 100, the number of training epochs is 300, the learning rate is 0.0001, and $\eta = 0.2$. For FM, the batch size is 100, the number of training epochs is 100, the learning rate is 0.0001, $\beta = 0$, and $\eta = 0.2$. We use Adam optimizer. Validation sets are used to determine optimal models during training.

C.4.2 Section 5.5.1.2

Dataset: We use CMU motion capture dataset. The training, validation, and test data are 10,000, 2,000, and 2,000, respectively.

Preprocessing: In the pre-processing step, the position and orientation of human body center (root) are removed. In addition, the joint angles that have nearly no movements (clavicles, fingers) are removed as done in FMVAE [33], and as a result, each pose data is expressed as a 50-dimensional vector.

Network Architecture: For VAE, FMVAE, and IRVAE, we use fully-connected neural networks that have two hidden layers with ReLU activation functions (512 nodes for each layer are used) for encoder and decoder. The output activation functions are linear and tanh for encoder and decoder, respectively. For FM, we use the RealNVP model of depth 8 and length 512.

Other Details: For VAE, FMVAE and IRVAE, the batch size is 100, the number of training epochs is 300, the learning rate is 0.0001. $\eta = 0.2$ for FMVAE and IRVAE. For FM, the batch size is 100, the number of training epochs is 300, the learning rate is 0.0001, $\beta = 0$, and $\eta = 0.2$. We use Adam optimizer. Validation sets are used to determine optimal models during training.

C.4.3 Section 5.5.2

Dataset: The number of training, validation, and test data are 162700, 19937, and 19962.

Network Architecture: We denote a Con2d layer by Con2d (input channel, output channel, kernel size, stride, padding) and ConvTranspose2d layer by ConvT2d (input channel, output channel, kernel size, stride, padding).

For VAE, FMVAE, and IRVAE, encoders are Convolutional Neural Networks with the following architecture: i) Conv2d (3, 128, 5, 2, 0), ii) Conv2d (128, 256, 5, 2, 0), iii) Conv2d (256, 512, 5, 2, 0), iv) Conv2d (512, 1024, 5, 2, 0), v) Conv2d (1024, 256, 1, 1, 0) with ReLU hidden layer activation functions, and decoders are Convolutional Neural Networks with the following architecture: i) ConvT2d (128, 1024, 8, 1, 0), ii) ConvT2d (1024, 512, 4, 2, 1), iii) ConvT2d (512, 256, 4, 2, 1), iv) ConvT2d (256, 128, 4, 2, 1) , and v) ConvT2d (128, 3, 1, 1, 0) with ReLU hidden layer activation functions. The output activation functions are linaer and tanh for encoder and decoder, respectively.

For FM, we use the RealNVP model of depth 8 and length 512.

For BR, we use the same encoder network as VAE, but add a normalizing layer (i.e., divide the output by its norm) at last (this addition improves the retrieval performance, very significantly).

For ResNet-50, we use the pre-trained model on ImageNet.

Other Details: For FMVAE and IRVAE, the batch size is 100, the number of training epochs is 100, the learning rate is 0.0001, and $\eta = 0.2$. For FM, the batch size is 100, the number of training epochs is 100, the learning rate is 0.00001, and $\eta = 0.2$. We use Adam optimizer. For BR, the batch size is 100, the number of training epochs is 100, the learning rate is 0.001 (binary cross entropy loss is used). Validation sets are used to determine optimal models during training.

For FMVAE, we test the following regularization coefficients $\alpha \in \{0.1, 1, 10, 100\}$ and report the best results at $\alpha = 1$. For IRVAE, we test the following regularization coefficients $\alpha \in \{1, 10, 100, 1000\}$ and report the best results at $\alpha = 10$. For FM, we use the stabilizing term with $\beta = \{10, 100\}$ and report the best results at $\beta = 100$; and it is empirically observed that using small learning rate is very important.

Cosine similarity is used as a similarity measure except for the ResNet-50 that uses Euclidean metric (we have tested both cosine similarity and Euclidean metric for each learned representation, and found that cosine similarity mostly works better except the ResNet-50).

C.5 Additional Experimental Results

C.5.1 Advantages of Isometric Regularization from a Generative Perspective

Autoencoders are not only useful for representation learning, but also can be used for realistic data generation. In this section, we show advantages of our isometric regularization from a generative perspective with the following experiments: (i) modulation in the latent space and (ii) linear interpolation in the latent space. We use the CMU motion capture data with the fully connected neural network as described in C.4.2. We use the latent space of dimension 8.

Latent Space Modulation : Figure C.1 shows eight generated poses for VAE and IRVAE obtained by the following procedure: (i) encode the original pose in the latent space, (ii) translate the encoded latent value along each latent space axis, and (iii) decode the translated values back to the data space. In case of VAE, translations along the latent space axes except z_2, z_6 axes do not generate new poses. In particular, the pose changes really a lot along the z_2 axis and yet just a little along the z_6 axis. This shows that the trained decoder in VAE is far from the isometry. In contrast, with the isometric regularization in IRVAE, each translated latent value along each latent space axis generates a different pose. Unlike VAE, diverse poses are generated, and in other words, a more disentangled representation is obtained. This property of the isometric decoder has a great advantage [115, 131].

Latent Space Interpolation : Figure C.2 shows sequences of poses generated from the latent space linear interpolants between a walking pose and balancing pose. In case of VAE, punching poses suddenly appear in the interpolation of the walking and balancing poses. In contrast, IRVAE does not produce such poses irrelevant to the two given start and end poses. The IRVAE produces more smoothly varying poses than VAE, and the pose interpolation obtained by the IRVAE is closer to the geodesic

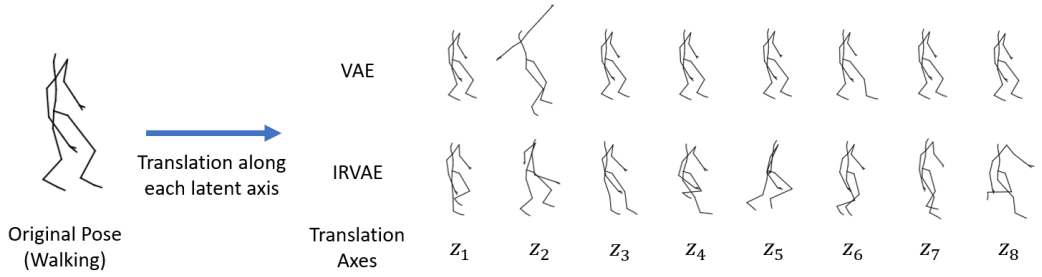


Figure C.1: The original pose is encoded in the latent space, then the encoded latent value is translated along each latent space axis ($z_1, z_2, ..., z_8$). The translated latent values are decoded back to generate a new eight pose for each model VAE and IRVAE. The translated distances are proportional to the standard deviations of the encoded training data.

interpolation along the pose data manifold.

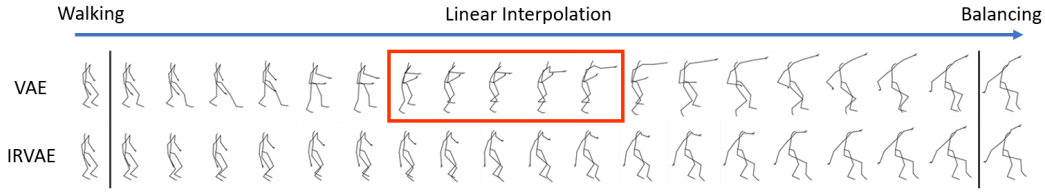


Figure C.2: Pose interpolations between a walking pose and balancing pose with linear interpolations in the latent spaces. The red box indicates suddenly appeared punching poses.

C.5.2 Diverse Image Data with Higher Latent Space Dimensions

We provide additional results with diverse image data (*MNIST, FMNIST, SVHN, CIFAR-10*) using various latent space dimensions. Figure C.3 shows that i) the tradeoff curves of IRVAE are located lower than those of FMVAE and ii) the FM learns more isometric representations without losses in MSEs.

Dataset: The number of training, validation, test data is 50,000, 10,000, 10,000 for MNIST, FMNIST, SVHN, and 45000, 5000, 10,000 for CIFAR-10.

Network Architecture: Let m denotes the latent space dimension. For VAE, FMVAE, IRVAE, encoders are Convolutional Neural Networks with the following architecture: i) Conv2d (3, 32, 4, 2, 0), ii) Conv2d (32, 64, 4, 2, 0), iii) Conv2d (64, 128, 4, 2, 0), iv) Conv2d (128, 256, 2, 2, 0), v) Conv2d (256, 2m, 1, 1, 0) with ReLU hidden layer activation functions, and decoders are Convolutional Neural Networks with the following architecture: i) ConvT2d (m, 256, 8, 1, 0) ii) ConvT2d (256, 128, 4, 2, 1) iii) ConvT2d (128, 64, 4, 2, 1) iv) ConvT2d (64, 3, 1, 1, 0) with ReLU hidden layer activation functions. The output activation functions are linear and sigmoid for encoder and decoder, respectively.

For FM, we use the RealNVP model of depth 8 and length 512.

Other Details: For FMVAE and IRVAE, the batch size is 100, the number of training epochs is 100, the learning rate is 0.0001, and $\eta = 0.2$. For FM, the batch size is 100, the number of training epochs is 100, the learning rate is 0.00001, and $\eta = 0.2$. We use Adam optimizer. Validation sets are used to determine optimal models during training.

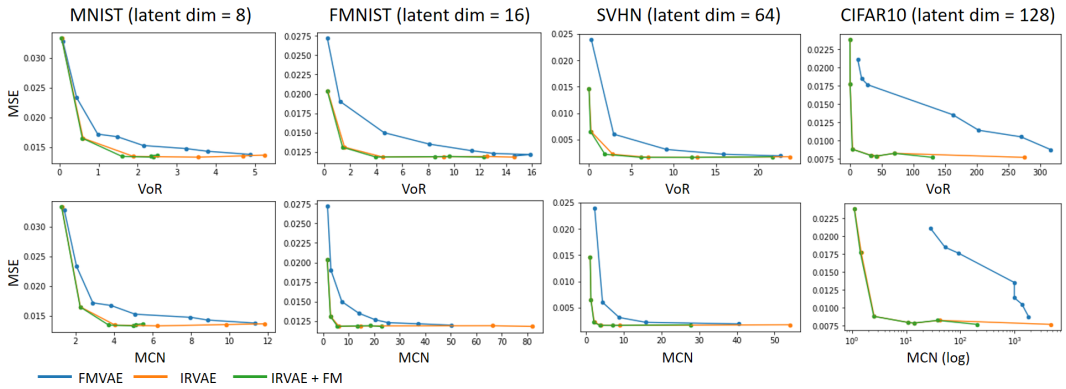


Figure C.3: The tradeoff curves of FMVAE, IRVAE, and IRVAE + FM.

C.5.3 Ablation Study on Mixup Parameter η

In the isometric regularization (and flattening module), we use the augmented encoded data distribution P_Z whose sampling procedure is given as follows: (i) encode data x_i, x_j to z_i, z_j by the encoder (and by the invertible map composed with the encoder) and (ii) compute a new latent value z with $z = \delta z_i + (1-\delta) z_j$ for δ uniformly sampled form $[-\eta, 1+\eta]$, where η is the mixup parameter. Then, our relaxed distortion measure

is defined over the support of P_Z, i.e.,

$$\mathbb{E}_{z\sim P_Z}\big[\sum_i (\frac{\lambda_i(z)}{\frac{1}{m}\mathbb{E}_{z\sim P_Z}[\sum_i \lambda_i(z)]} - 1)^2\big],$$

where f is the decoder, J_f is the Jacobian of f, and $\lambda_i(z)$ are eigenvalues of $J_f^T J_f(z)$. The mixup augmentation technique is used to extend the influence of isometric regularization to an area where data does not exist.

In this section, we visually analyze the effects of the mixup augmentations in a range of mixup parameters η. We use the CMU data with fully connected networks described in C.4.2 and 2-dimensional latent spaces.

Figure C.4 shows the latent space data distributions with some equidistance ellipses (the colors of the ellipses represent the condition numbers; redder-the-bigger). In no mixup case, there are many un-isotropic red ellipses in regions between data points, meaning that they are not isometrically regularized. Mixup augmentation makes it possible to regularize even regions between data. When the mixup parameter $\eta = 0$, only the area covered by data interpolation is regularized. By increasing the mixup parameter η, the outer area, which is covered by data extrapolation, begins to be regularized. Although using bigger eta seems to be unconditionally good because the wider area is regularized, there is indeed a tradeoff as we can see from Figure C.4. When the mixup parameter is too big as $\eta = 1$, some inner area (i.e., area covered by data interpolation) is not sufficiently regularized because of the cost of regularizing the outer domain. We use a balanced value $\eta = 0.2$ for all experiments conducted in this thesis (for all algorithms IRVAE, FM and FMVAE).

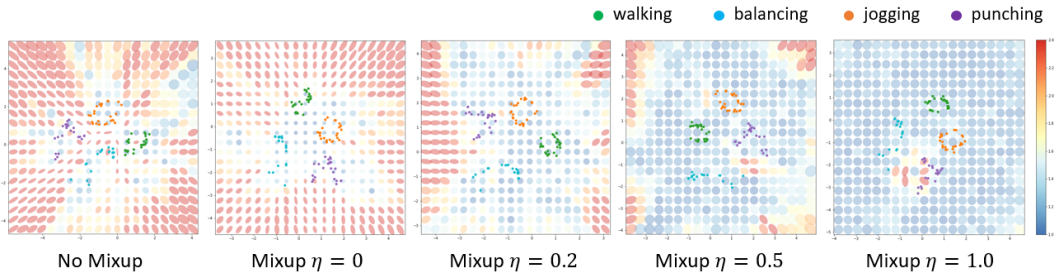


Figure C.4: The effect of mixup augmentation with varying mixup parameters η.

C.5.4 Computational Speed

In this section, we provide per-epoch runtimes of VAE, FMVAE, IRVAE, and FM. We use the MNIST data and same experiment settings in C.4.1. We use the GeForce RTX 3090 for GPU resources. Table C.1 shows the per-epoch runtimes. The difference in computation times arises from the computations of the regularization terms. The IRVAE takes a little longer than the FMVAE because, while the regularization term in FMVAE requires to compute the Jacobian vector product only, the regularization term in IRVAE requires to compute both the Jacobian vector product and vector Jacobian product. In the case of FM, it takes much longer because the invertible neural network architecture is relatively heavier. Fortunately, in training of FM that uses the pre-trained IRVAE, we empirically observed that only a few epochs are required for convergence, probably because the IRVAE already had learned isometric representations to a certain degree.

Table C.1: Averages and standard deviations of the per-epoch runtimes. 50,000 MNIST image training data are used with 100 batch size. For VAE, FMVAE, IRVAE, we use the 4 layers of fully connected neural networks, and for FM, we use the RealNVP of depth 8.

	VAE	FMVAE	IRVAE	FM
per-epoch runtime	1.91 $\pm$ 0.0818 s	2.80 $\pm$ 0.0755 s	2.81 $\pm$ 0.0869 s	19.9 $\pm$ 0.235 s

C.5.5 Isometric Regularization for Other Autoencoders

Our isometric regularization algorithm can be used with other types of autoencoders although we focus on the Variational Autoencoder (VAE) in the main manuscript. In this section, we verify the effects of the isometric regularization with more diverse autoencoder methods: Vanilla Autoencoder (AE) [9], Wasserstein Autoencoder (WAE) [40], and Denoising Autoencoder (DAE) [18].

We use the MNIST data and same experiment settings in C.4.1. For DAE, we use the Gaussian noise with a standard deviation of 0.01. For WAE, we use the maximum mean discrepancy, median heuristic for bandwidth selection, and the regularization coefficient 0.001.

Figure C.5 shows the effects of the isometric regularization to diverse autoncoders (AE, VAE, DAE, WAE). Regardless of the autoencoder types, we consistently observe that the added isometric regularization terms effectively separate data of different classes (even without major tuning of the regularization coefficient α).

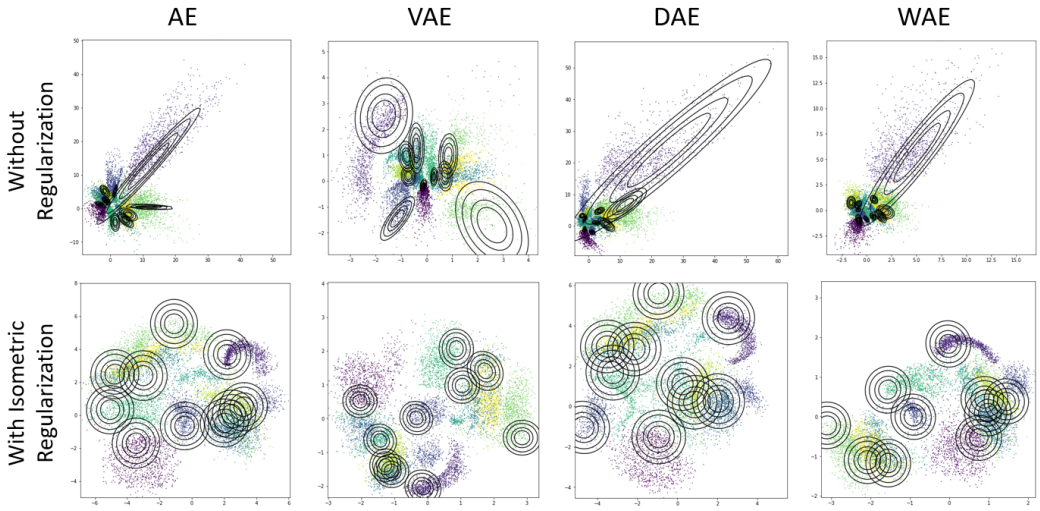


Figure C.5: Isometric regularization with diverse autoencoder methods. The more homogeneous and isotropic equidistance plots are, the more isometric the representations are.

C.5.6 Isometry vs Scaled Isometry

In this section, we show empirical results that support our claim in the introduction that finding a scaled isometry is better than finding a strict isometry.

Whereas an isometry exactly preserves angles and distances, a scaled isometry preserves angles and scaled distances. At first this may seem a superficial difference – why not simply choose length scales so that the two spaces have the same scales? – but the reason this difference is consequential for our problem is that we do not a priori have a precise charcterization (and hence their relative scales) of the two spaces. We are in effect discovering the manifold structure of the data space while constructing a latent space representation for the data manifold at the same time.

To characterize the above more precisely, let $f(z)$ be a decoder, $J_f(z)$ be the Jacobian of $f(z)$, and $P_Z(z)$ be the augmented encoded data distribution. Let $\lambda_i(z)$ be eigenvalues of $J_f^T(z)J_f(z)$. Let $G(z)$ be the latent space metric, and $H(x) = I$. Recall that the isometry f needs to satisfy $J_f^T(z)J_f(z) = G(z)$ for all z in the support of P_Z, whereas the scaled isometry f needs to satisfy $J_f^T(z)J_f(z) = cG(z)$ for some $c > 0$ and all z in the support of P_Z. To find a strict isometry, we need to set a scalar parameter k that determines the latent space metric, i.e., $G(z) = kI$, in advance, whereas, to find a scaled isometry, we just need to set $G(z) = I$ as done in the

thesis.

To find the isometry, we add the following distortion measure as a regularization term:

$$\alpha \mathbb{E}_{z\sim P_Z}[\sum_i (\lambda_i(z) - k)^2],$$

where k is the pre-defined parameter and α is the regularization coefficient. We need to search for the optimal parameter k in this case.

In contrast, we do not need such parameter k to find a scaled isometry. We just need to add the following relaxed distortion measure as a regularization term:

$$\alpha \mathbb{E}_{z\sim P_Z}\big[\sum_i (\frac{\lambda_i(z)}{\frac{1}{m}\mathbb{E}_{z\sim P_Z}[\sum_i \lambda_i(z)]} - 1)^2\big],$$

where α is the regularization coefficient.

Figure C.6 shows the tradeoff curves (lower-the-better) obtained by using (i) the relaxed distortion measure (for finding a scaled isometry) or (ii) the distortion measure with a range of user-defined parameters k (for finding a strict isometry). For the strict isometry learning case, it can be seen that the performance peaks when the value of k is at an appropriate level that is neither too high nor too low. In this experiment, the optimal k lies in between 10 and 1000 and can be experimentally found only by performing a finer grid search. The tradeoff curves obtained by using the relaxed distortion measure are always located lower than the tradeoff curves obtained by using the distortion measures. From this experimental result, we conclude that it is better to find a scaled isometry than strict isometry.

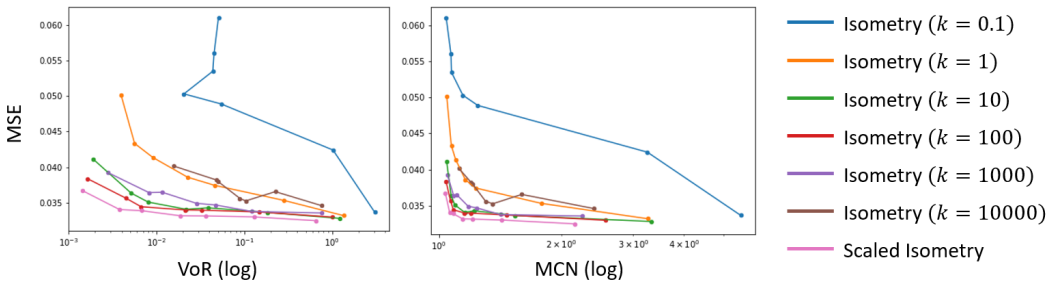


Figure C.6: Tradeoff curves obtained by changing the regularization coefficients α (lower-the-better).

C.5.7 Multiple Times Run to compute Averages and Standard Deviations

It is important to judge whether the experimental result is a result of randomness or not. In this section, we take the same experiments done in Figure 1 and 3 of the main manuscript with MNIST and CMU data multiple times, and report the averaged tradeoff curves and standard deviations of the measured metrics.

Figure C.7 shows the results, where the standard deviations computed over multiple experiments with different random seeds are visualized as ellipses. The horizontal axes lengths of the ellipses represent the standard deviations of the VoR and MCN, and the vertical axes lengths of the ellipses represent the standard deviations of the MSE. It is correct to use the standard errors to see the standard deviations of the sample means, i.e., standard deviations divided by the square root of the number of experiments, but we use the standard deviations since the standard errors are too small to visualize. The sizes of the ellipses are rally small, which provides a solid evidence that IRVAE outperforms FMVAE.

C.5.8 Sensitivity Analysis of the Latent Space Dimension

In general, when training an autoencoder, the most important parameter is the latent space dimension. Although there have been some works on estimating the intrinsic dimension of the data distribution [132], given a complex high-dimensional data, the intrinsic dimension estimation problem is very challenging; in practice, an appropriate value is often found, empirically. Throughout literature, there are some frequently-used choices for well-known standard image datasets. For relatively simple datasets such the MNIST and FMNIST, latent space dimensions 2 $\sim$ 32 are often used, and for more complex and higher dimensional datasets such as the CIFAR10 and CELEBA, latent dimensions 64 $\sim$ 512 are often used.

In this section, using the MNIST image data with the fully connected network described in C.4.1 (trained for 100 epochs), we provide an analysis of the isometric regularization in autoencoders with multiple latent space dimensions. Table C.2 shows averages and standard deviations of the condition numbers of the pullback metrics $J_f^T(z)J_f(z)$ (f is the decoder and J_f is the Jacobian of f) computed with the test data. Both the averages and standard deviations should be small if $J_f^T(z)J_f(z) = cI$ for some $c > 0$ for all encoded test data z.

In the case of VAE, it can be seen that both the average and standard deviation of the condition number sharply increase as the latent dimension increases. On the other hand, IRVAE appears to be able to effectively find an isometric representation even if the dimension of the latent space increases. In the case of 16-dimensional latent

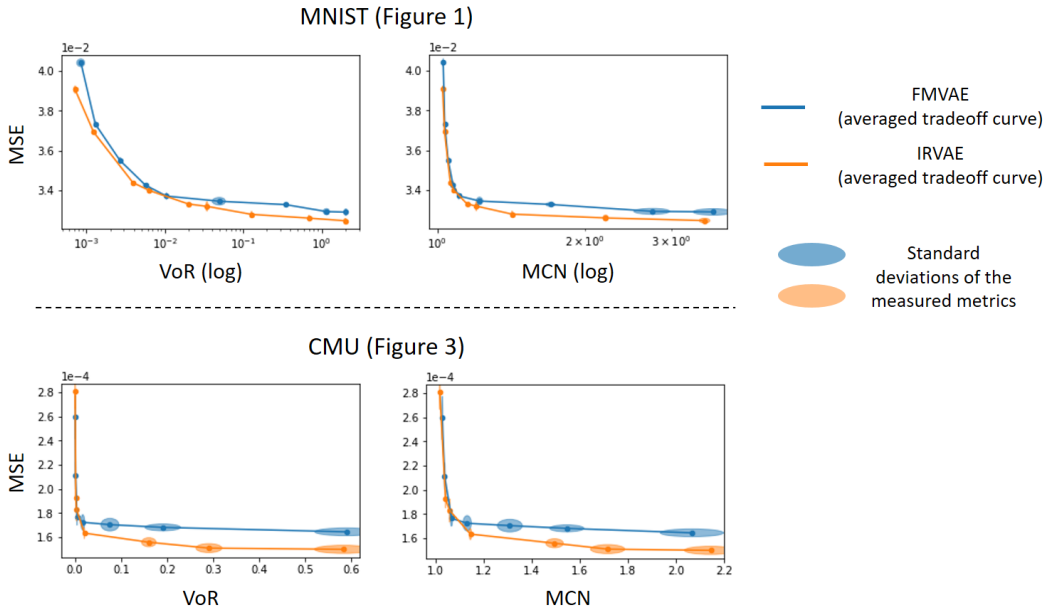


Figure C.7: Averaged tradeoff curves and standard deviations represented as ellipses for MNIST and CMU experiments in Figure 1 and 3 of the main manuscript (20 times run). We wanted to draw ellipses by using the standard errors, but they were too small to visualize. Even standard deviations are really small.

space, the condition number of IRVAE is about 32 which is not small enough, and this result may imply that the true intrinsic dimension of the MNIST data manifold is lower than 16.

In summary, our isometric regularization can effectively find more isometric representations than vanilla autoencoders regardless of the latent space dimensions. When the dimension of the latent space is too large, it may not be possible to obtain a sufficiently isometric representation space. This This seems because the dimension of the ground true manifold is much smaller than the selected latent space dimension.

C.6 Detailed Results for Single Attribute Retrieval

Table C.3 shows the detailed retrieval results (P@10 of the single attribute retrieval). Our algorithms (IRVAE and IRVAE + FM) have much higher performance than FMVAE and VAE, and even show performance close to the supervised method (BR).

Table C.2: Averages and standard deviations of the condition numbers of the pullback metrics for VAE and IRVAE with respect to the latent space dimension. The lower the average and standard deviations are, the more isometric the latent is.

Latent Space Dimension	VAE	IRVAE
2	5.39 ± 4.72	1.41 ± 0.55
8	85.55 ± 85.69	1.87 ± 0.25
16	1635018.88 ± 1789834.25	37.28 ± 42.12

Table C.3: Single attribute retrieval results of Precision at 10, P@10 (for total 40 attributes). The best results among unsupervised methods are colored red, and the best results among all six methods are marked bold.

Attributes	**VAE**	**FMVAE**	**IRVAE (ours)**	**IRVAE + FM (ours)**	**Pre-trained (ResNet-50)**	**Supervised (BR)**
Young	0.8	**1**	0.7	0.8	0.7	**1**
Wearing Necktie	0.4	**0.7**	0.6	**0.7**	0.1	0.5
Wearing Necklace	0.2	0.1	**0.3**	0.2	0.1	0.1
Wearing Lipstick	**1**	0.9	**1**	**1**	0.4	**1**
Wearing Hat	0.4	0.7	0.8	0.9	0	**1**
Wearing Earrings	0.5	0.5	0.7	0.6	0.4	**1**
Wavy Hair	0.8	**0.9**	**0.9**	**0.9**	0.3	**0.9**
Straight Hair	**0.9**	0.2	0.7	0.7	0.2	0.5
Smiling	**1**	**1**	**1**	**1**	0.7	**1**
Sideburns	0.4	0.7	**1**	0.9	0.2	0.6
Rosy Cheeks	0.6	0.5	0.8	0.7	0	**1**
Receding Hairline	0.3	**0.8**	0.6	0.5	0	**0.8**
Pointy Nose	0.5	0.6	0.4	**0.8**	0.2	**0.8**
Pale Skin	0.3	**0.8**	0.7	0.7	0.1	**0.8**
Oval Face	0.5	**0.6**	0.5	0.4	0.5	**0.6**
No Beard	0.8	**1**	**1**	**1**	0.7	**1**
Narrow Eyes	0.5	0.1	0.3	0.5	0.1	**0.7**
Mustache	0.2	0.3	0.3	**0.4**	0.1	0.3
Mouth Slightly Open	**1**	**1**	**1**	**1**	0.6	**1**
Male	**1**	**1**	**1**	**1**	0.4	**1**
High Cheekbones	**1**	**1**	0.9	**1**	0.4	**1**
Heavy Makeup	**1**	0.8	**1**	**1**	0.3	**1**
Gray Hair	0.5	0.7	0.7	**0.8**	0	0.7
Goatee	0.5	0.6	0.7	**0.9**	0	0.7
Eyeglasses	**1**	0.9	**1**	**1**	0.1	**1**
Double Chin	0.4	0.3	0.6	0.8	0.1	**1**
Chubby	0.3	0.1	0.3	0.4	0	**0.9**
Bushy Eyebrows	0.6	0.5	**1**	**1**	0.3	0.9
Brown Hair	0.5	0.3	**0.8**	0.6	0.2	**0.8**
Blurry	0	0.1	0.2	0.1	0.1	**0.4**
Blond Hair	**1**	**1**	**1**	**1**	0.1	0.9
Black Hair	0.4	0.9	0.9	**1**	0.2	0.8
Big Nose	0.4	**0.6**	0.5	0.5	0.1	**0.6**
Big Lips	0.4	**0.6**	0.2	0.5	0.3	0.3
Bangs	**1**	0.8	**1**	**1**	0	**1**
Bald	0.3	0.4	0.6	0.8	0	**0.9**
Bags Under Eyes	0.5	0.4	0.6	**0.7**	0.4	**0.7**
Attractive	**1**	**1**	0.9	0.9	0.5	**1**
Arched Eyebrows	0.8	0.7	0.7	0.8	0.2	**0.9**
5 o'clock Shadow	**0.7**	0.5	**0.7**	0.6	0.3	1
Number of REDS	14	15	18	25	-	-
Number of BOLDS	12	15	15	19	0	28
Average Precision	61.0	64.0	71.5	75.3	23.5	80.3

D

Appendix: A Statistical Manifold Framework for Point Cloud Data

D.1 Existing Geometric/Statistical Methods for Point Cloud Data

D.1.1 Geometric Methods

The Hausdorff distance measures the distance between two non-empty subsets of a metric space [66, 133]. Given two point clouds $\mathbf{X} = \{x_1, ..., x_n \mid x_i \in \mathbb{R}^D\}$ and $\mathbf{Y} = \{y_1, ..., y_n \mid y_i \in \mathbb{R}^D\}$ and metric $\|x - y\|^2$ in $\mathbb{R}^D$, the Hausdorff distance can be computed as follows:

$$\max(\max_{x \in \mathbf{X}}(\min_{y \in \mathbf{Y}} \|x - y\|^2), \max_{y \in \mathbf{Y}}(\min_{x \in \mathbf{X}} \|x - y\|^2)).$$

The Hausdorff distance is susceptible to outliers; hence, in practice, the average Hausdorff distance is used more often:

$$\frac{1}{|\mathbf{X}|} \sum_{x \in \mathbf{X}} \min_{y \in \mathbf{Y}} \|x - y\|^2 + \frac{1}{|\mathbf{Y}|} \sum_{y \in \mathbf{Y}} \min_{x \in \mathbf{X}} \|x - y\|^2,$$

where $|\mathbf{X}|$ denotes the number of elements in the set $\mathbf{X}$. A slightly modified version of this is often referred to as the Chamfer distance [93]. The popular point cloud registration algorithm ICP relies on these classes of metrics [134].

Another popular similarity measure between two point cloud data is the Earth Mover's Distance (EMD) [67]:

$$\sum_{x \in \mathbf{X}} \min_{\phi : \mathbf{X} \to \mathbf{Y}} \|x - \phi(x)\|^2,$$

where ϕ is a bijective mapping. Although the EMD is computationally more expensive than the above Hausdorff distances, comparing point clouds with optimal matching in EMD provides a more robust and well-behaved similarity measure.

The Chamfer distance and EMD are often used to measure the distances between two point clouds, but they are typically computationally expensive. Recently, the sliced Wasserstein distance and its variants have been proposed to more efficiently measure distances [135]. In another study, each point cloud data is represented as a matrix of pairwise Euclidean distances between all points, and the Frobenius norm of the difference between the two matrices is used as the distance between two point clouds [136].

D.1.2 Statistical Methods

Interpreting the point cloud data as a set of samples from some underlying probability distribution is very intuitive and natural, and has been adopted in many previous works [70, 71, 72, 73, 74, 75, 76]. Commonly, a mixture model is used to describe the point cloud, written as follows:

$$p(x; w, \theta) := \sum_{i=1}^{k} w_i \phi(x|\theta_i),$$

where the w_i are weights and $\phi(x|\theta_i)$ are primitive density functions with parameter θ_i (e.g., $\phi(x|\theta_i)$ can be a standard Gaussian with mean θ_i). Given a point cloud $\mathbf{X} := \{x_1, \ldots, x_n | x_i \in \mathbb{R}^D\}$, the parameters w, θ are either fit with data or specified by the user, then the mixture model is used as a statistical representation of the point cloud data. Besides the most popular choice for ϕ, the Gaussian [70], other choices such as the t-distribution [74] or hybrid model [75] have been explored. The main purpose behind using statistical representations in existing works are to use the well-known information-theoretic divergence measures such as the KL-divergence to compute the similarity between point clouds.

However, all these methods focus on distance metrics that measure just one aspect of point cloud data, yet effective mathematical concepts and tools for defining and measuring other important geometric aspects of point cloud are still lacking.

D.2 Proof of the Propositions

D.2.1 Proof of Proposition 6.2.2

Proof. Start with the proof of Proposition D.2.1 which is an easier version of the original proposition.

Proposition D.2.1. Assume that all point cloud data in $\mathcal{X}$ consists of exactly n distinct points in $\mathbb{R}^D$. If the set of functions $\{K(\Sigma^{-\frac{1}{2}}(x-x_i))\}_{x_i\in\mathcal{F}}$ are linearly independent[1] for any arbitrary finite subset $\mathcal{F}\subset\mathbb{R}^D$ with $|\mathcal{F}|\leq 2n$, the mapping $h:\mathcal{X}\to\mathcal{S}$ is 1-1.

Proof. Let's consider two point clouds $\{y_i\}_{i=1}^n$ and $\{z_i\}_{i=1}^n$. To show that the mapping $h:\mathcal{X}\to\mathcal{S}$ is 1-1 (especially injective), we have to prove the following statement:

$$p(x;\{y_i\}_{i=1}^n)=p(x;\{z_i\}_{i=1}^n)\implies\{y_i\}_{i=1}^n=\{z_i\}_{i=1}^n. \tag{D.2.1}$$

The conditional statement can be rewritten as follows:

$$\sum_{i=1}^n K(\Sigma^{-\frac{1}{2}}(x-y_i))=\sum_{i=1}^n K(\Sigma^{-\frac{1}{2}}(x-z_i)). \tag{D.2.2}$$

Let denote $B=\{y_i\}_{i=1}^n\cap\{z_i\}_{i=1}^n$ and $|B|=m$, and assume that $m<n$. Then the above equation is reduced to

$$\sum_{y\in\{y_i\}_{i=1}^n-B}K(\Sigma^{-\frac{1}{2}}(x-y))-\sum_{z\in\{z_i\}_{i=1}^n-B}K(\Sigma^{-\frac{1}{2}}(x-z))=0. \tag{D.2.3}$$

Since the sets $\{y_i\}_{i=1}^n-B$ and $\{z_i\}_{i=1}^n-B$ are disjoint and each set has $n-m$ elements, the LHS has $2(n-m)\leq 2n$ terms and the terms are different to each other. Then, by the assumption, the above $2(n-m)$ terms are linearly independent, and so the above equation cannot hold. There is a contradiction and the assumption $m<n$ must be wrong. Therefore, $m=n$, so $\{y_i\}_{i=1}^n=\{z_i\}_{i=1}^n$. □

With the above proposition, any kernel function that satisfies the linear independence condition is sufficient to ensure the existence of a 1-1 mapping h. Indeed, the strict positive definiteness of the kernel function, satisfied by various kernel functions such as normal (Gaussian) or Laplacian function, implies the linear independence condition as stated below:

Proposition D.2.2 (Corollary of Proposition 4.3. in [137])**.** Let $\Psi:\mathbb{R}^D\times\mathbb{R}^D\to\mathbb{R}$ be a positive function and $\mathcal{F}=\{x_1,...,x_n\}$ be a finite set of mutually distinct points. Then the set $\{\Psi(\cdot,x_i)\}_{x_i\in\mathcal{F}}$ is linearly independent if and only if the matrix $(\Psi(x_i,x_j))_{i,j=1,...,n}$ is positive definite.

[1]The linear independence of a set of functions implies that only a trivial linear combination of the functions equals the zero function.

From Propositions D.2.1 and D.2.2, Proposition 6.2.2 can be easily proved. From the assumption that a function $\Psi : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ defined by

$$\Psi(x, y) = K(\Sigma^{-\frac{1}{2}}(x - y)) \tag{D.2.4}$$

is strictly positive definite, the matrix $(\Psi(x_i, x_j))_{i,j=1,...,n}$ is always positive definite for any finite set of mutually distinct points $\mathcal{F} = \{x_1, ..., x_n\}$, so the set of functions $\{K(\Sigma^{-\frac{1}{2}}(x - x_i)\}_{x_i \in \mathcal{F}} = \{\Psi(x, x_i)\}_{x_i \in \mathcal{F}}$ is always linearly independent by Proposition D.2.2. Then, directly from Proposition D.2.1, the mapping $h : \mathcal{X} \to \mathcal{S}$ is 1-1.

□

D.2.2 Proof of Proposition 6.2.3

Proof. Since $\frac{\partial \log p(x;X)}{\partial X^{ij}} = \frac{1}{p(x;X)} \frac{\partial p(x;X)}{\partial X^{ij}}$, the Riemannian metric H_{ijkl} in equation (6.2.7) is

$$\int p(x; X) \frac{1}{p^2(x; X)} \frac{\partial p(x; X)}{\partial X^{ij}} \frac{\partial p(x; X)}{\partial X^{kl}} \, dx.$$

By plugging $p(x; X)$ in equation (6.2.6) in $\frac{\partial p(x;X))}{\partial X^{ij}}$, we get the following expression:

$$\begin{aligned}
n\sqrt{|\Sigma|} \, \frac{\partial p(x; X)}{\partial X^{ij}} &= \frac{\partial}{\partial X^{ij}} \sum_{a=1}^{n} K(\Sigma^{-1/2}(x - x_a)) \\
&= \frac{\partial}{\partial X^{ij}} K(\Sigma^{-1/2}(x - x_i)) \\
&= J_K(\Sigma^{-1/2}(x - x_i))\Sigma^{-1/2} \frac{\partial}{\partial X^{ij}}(x - x_i) \\
&= J_K(\Sigma^{-1/2}(x - x_i))\Sigma^{-1/2} \frac{\partial}{\partial X^{ij}}(x - x_i) \\
&= -[J_K(\Sigma^{-1/2}(x - x_i))\Sigma^{-1/2}]_j,
\end{aligned}$$

$J_K : \mathbb{R}^D \to \mathbb{R}^D$ is the Jacobian of the kernel function K. This consequently leads to

$$H_{ijkl}(X) := \int p(x; X) \frac{(J_K\big|_{h(x,x_i)} \Sigma^{-\frac{1}{2}})_j (J_K\big|_{h(x,x_k)} \Sigma^{-\frac{1}{2}})_l}{(\sum_{m=1}^{n} K(h(x, x_m)))^2} \, dx,$$

where $h(x, x_i) = \Sigma^{-\frac{1}{2}}(x - x_i)$.

If K is the standard normal kernel function, then we get $J_K(x) = -K(x)x^T$. By

plugging this in the above equation with $\Sigma = \sigma^2 I$, we get the following:

$$H_{ijkl}(X) = \int p(x;X) \frac{(J_K|_{h(x,x_i)} \Sigma^{-\frac{1}{2}})_j (J_K|_{h(x,x_k)} \Sigma^{-\frac{1}{2}})_l}{(\sum_{m=1}^n K(h(x,x_m)))^2} \, dx$$
$$= \int p(x;X) \frac{K(h(x,x_i))K(h(x,x_k))}{(\sum_{m=1}^n K(h(x,x_m)))^2} \Big[\frac{(x-x_i)(x-x_k)^T}{\sigma^4} \Big]_{jl} \, dx.$$

□

D.3 Implementation Details for the Experiments

D.3.1 Synthetic 3D Basic Shape Dataset Generation

For synthetic 3D basic shape dataset, we define 5 shape classes that consist of cylinder, cone, elliptic cone, ellipsoid, and box. Figure D.1 shows the representative shape of each class and the shape parameters used to define the shape class. We sample 512 points from the surface mesh of the shapes using a greedy sample elimination algorithm, and each point cloud is then normalized so that the two farthest points are a unit distance apart.

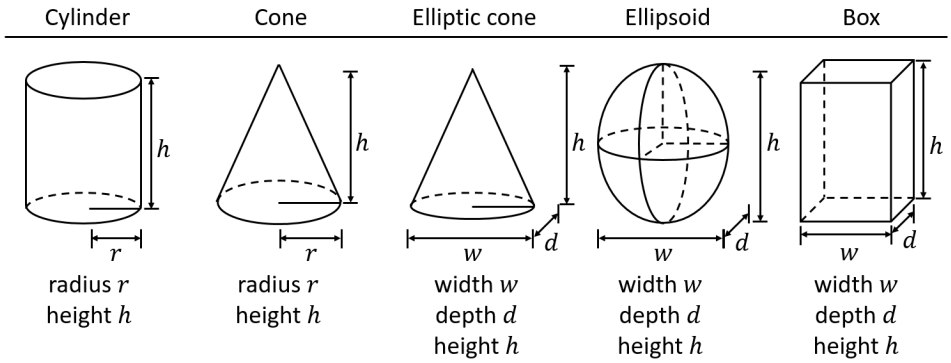


Figure D.1: Representative shape to each class and the shape parameters required to define it. There are 5 shape classes including cylinder, cone, elliptic cone, ellipsoid, and box.

In Section 6.4.1.1, we use a dataset consisting of cones, cylinders, and ellipsoids, which are split into training/validation/test sets of size 3196/800/804. The detail ranges

of the exact value of the shape parameters are shown in Table D.1.

Table D.1: The ranges of the shape parameters of the dataset

SHAPE	param	min	max	param	min	max	param	min	max
Cylinder	r	0.01	0.12	h	0.05	0.45			
Cone	r	0.02	0.15	h	0.02	0.45			
Ellipsoid	w	0.03	0.12	d	0.03	0.12	h	0.03	0.12

In Section 6.4.1.2, we use a dataset consisting of boxes, cones, and ellipsoids divided into training/validation/test sets of size 720/240/240. The detail ranges of the aspect ratios of the shape parameters are shown in Table D.2.

Table D.2: The ranges of the shape parameters of the dataset

SHAPE	param	min	max	param	min	max
Elliptic cone	d/w	0.33	3	h/w	0.33	3
Ellipsoid	d/w	0.33	3	h/w	0.125	0.33
Box	d/w	0.33	3	h/w	0.33	3

D.3.2 Details for Experiments on Synthetic 3D Basic Shape Dataset

We used an encoder with a structure similar to the classification network used in DGCNN [98]. The input point cloud with dimension 3×512 passes through five EdgeConv layers with point-wise latent space dimensions (64, 64, 128, 256) and a max pooling layer (we do not use a batch normalization layer unlike the original DGCNN classification network, but other settings are the same, e.g., $k = 20$, leaky relu activation), then we can obtain a 1024-dimensional feature vector. Then this feature vector again passes through three fully-connected neural networks with dimensions (512, 256, 2) with leaky relu activation functions and linear output activation function; the latent space is two-dimensional. For the decoder model, we simply use a fully-connected neural network as the decoder. The two-dimensional vector on the latent space passes through three fully-connected neural networks with dimensions (256, 512, 3×512) with relu activation functions and linear output activation function; the output is a 3D point cloud with the number of points 512.

Section 6.4.1.1: To train the networks, we use ADAM with a learning rate of 0.001 and batch size of 16; the total number of the epochs is 500. The mean value of MEDs of the dataset is 0.0339, and we use the bandwidth value k to 0.5. We use

Chamfer distance as the reconstruction loss; for regularization figures, the regularization term with the version of Equation (6.3.12) is multiplied by a coefficient $\lambda = 10^7$ with the info-Riemannian metric and by a coefficient $\lambda = 1$ with the Euclidean metric and added to the reconstruction loss term for each metric case. The value of η is set to be 0.0. For geodesic computation, we parametrize the curve $z(t)$ by a cubic spline with fixed boundary points z_1, z_2 and 10 control points. The control points are first initialized with equally spaced linear interpolants between z_1 and z_2. Then, for each iteration of optimization, we randomly sample 40 points on $t_i \sim U(0,1), i = 1, ..., 40$ and calculate an expectation $\frac{1}{40}\sum_{i=1}^{40} \dot{z}(t_i)^T G(z(t_i))\dot{z}(t_i)$ over the sampled points as the approximation of the objective function. We use ADAM with a learning rate of 0.001 and the total number of the iterations is 5000.

Section 6.4.1.2: To train the networks, we use ADAM with a learning rate of 0.001 and batch size of 16; the total number of the epochs is 3000. The mean value of MEDs of the dataset is 0.0341, and we use the bandwidth value k to 0.5. We use Chamfer distance as the reconstruction loss, and the regularization term with the version of Equation (6.3.12) is multiplied by a coefficient $\lambda = 10^7$ and added to the reconstruction loss term. The value of η is set to be 0.0.

To quantitatively evaluate how much the regularization approach improves class separability, more diverse synthetic datasets are made and experiments are conducted. We use datasets consisting of boxes, elliptic cones, and ellipsoids. In details, we generate short, normal, and tall shapes for each shape class, and the aspect ratios of the shape parameters are shown in Table D.3. We conduct a total of 27 experiments with 3^3 combinations. Each dataset is divided into training/validation/test sets of size 720/240/240. Training configurations are the same with the above experiment, except that the mean values of MEDs are different to each other (but we consistently use the bandwidth value k to 0.5) and the total number of epochs is 500.

Table D.3: The ranges of the shape parameters of the dataset used in quantitative analysis on synthetic dataset

SHAPE	param	min	max	param	min	max
Elliptic cone short	d/w	0.33	3	h/w	0.125	0.33
Elliptic cone normal	d/w	0.33	3	h/w	0.33	3
Elliptic cone tall	d/w	0.33	3	h/w	3	8
Ellipsoid short	d/w	0.33	3	h/w	0.125	0.33
Ellipsoid normal	d/w	0.33	3	h/w	0.33	3
Ellipsoid tall	d/w	0.33	3	h/w	3	8
Box short	d/w	0.33	3	h/w	0.125	0.33
Box normal	d/w	0.33	3	h/w	0.33	3
Box tall	d/w	0.33	3	h/w	3	8

In the obtained representation spaces, after fitting the Gaussian mixture model by using the training and validation data, the clustering scores are measured with the test data.

Color assigning method: To visually indicate which class generated point clouds belong to, we color these according to the ratio of the Chamfer distances to the nearest point cloud for each class. In detail, the smallest value (distance to nearest point cloud) is found by comparing the distance between the given point cloud and all point clouds of each class in the dataset. Since we are using 3 shape classes in both examples, we call the nearest distance to each class d_1, d_2, and d_3. After that, the vector $d = (d_1, d_2, d_3)$ is normalized with 2-norm so that the 2-norm of the vector to be 1. Finally, the value $0.2 \times \mathrm{Softmax}(1/d_1, 1/d_2, 1/d_3)$ is regarded as the ratio of the distances and a color is assigned to a given point cloud according to this ratio (i.e., linear weighted sum in the RGB coordinate).

D.3.3 Details for Experiments on Standard Benchmark Dataset

We use four different point cloud autoencoders: *FcNet*, *FoldingNet*, *PointCapsNet*, and *DGCNN-FcNet*; the latent space is 512-dimensional. For *FcNet* and *FoldingNet*, we use the exactly same point cloud autoencoder structures both adopted from [93]. For *PointCapsNet*, we also use the exactly same point cloud autoencoder structure adopted from [99]; we use 16×32 capsules to restrict the latent space to a reasonable size of 512. For *DGCNN-FcNet*, we use DGCNN classification network as encoder (i.e., the same encoder architecture used in experiments on synthetic 3D basic shape dataset, see Appendix D.3.2), and the same decoder structure from *FcNet* as decoder (i.e., three fully-connected neural networks with dimension (1024, 2048, 3×2048) with relu activation function and linear output activation function). To train the networks, we use ADAM with a learning rate of 0.0001, betas of $[0.9, 0.999]$, and weight decay of 0.000001 and batch size of 16; the total number of the epochs is 500. The mean value of MEDs of the dataset is 0.0356, and we use the bandwidth value k to 0.8. We use Chamfer distance as the reconstruction loss and regularization term with the version of Equation (6.3.13) with the value of η to be 0.2. The regularization term is multiplied by various coefficients, where the values of the regularization coefficients are summarized in Appendix D.3.

D.3.4 Details for Experiments on Standard Benchmark Dataset with Noise

We use the exactly same point cloud autoencoder structures adopted from [93], *FcNet*, where the latent space is 512-dimensional. We add noise to each point $\mathbf{x}$ in point cloud of the dataset (ShapeNet, ModelNet10, and ModelNet40) according to $\mathbf{x} \mapsto \mathbf{x} + m\mathbf{v}$,

where **v** is uniformly sampled on the unit sphere and m is sampled from the Gaussian distribution with zero mean and different levels of standard deviation (1%, 5%, 10%, and 20% of the diagonal length of the point cloud bounding box) as done in [138]. The training configuration is the same with the case of Appendix D.3.3 except the followings. The regularization term is multiplied by $\lambda = 8000$. The mean values of MEDs of the dataset are 0.0320, 0.0364, 0.0442, and 0.579 for the cases of the noise levels 1%, 5%, 10%, and 20%, respectively, and we use the bandwidth value k to 0.8.

D.3.5 Details for Experiments on Standard Benchmark Dataset (Semi-Supervised Classification)

We train *FcNet* whose latent space is 512-dimensional with and without regularization. The training configuration is the same with the case of Appendix D.3.3 except the following: the regularization term of the regularized autoencoder (i.e., FcNet + I) is multiplied by $\lambda = 8000$. In this case, when we training linear SVM classifier, we use the different numbers of training data (1%, 5%, 10%, and 50% of the overall training data).

D.4 Additional Experimental Results

D.4.1 Synthetic Dataset

D.4.1.1 Qualitative Results of Table 6.1 in Section 6.4.1.2.

More examples related to the experiment in Section 6.4.1.2 are shown in Figure D.2. The trend of the experimental results is similar to the experimental results in Section 6.4.1.2. For all five results, the gray ellipses are aligned well with the decision boundary in the vanilla autoencoder (we show the decision boundary in Figure D.2 while it is not included in the main manuscript due to lack of space). In the regularized autoencoder, these gray ellipses (or Riemannian metrics) try to become isotropic, so the gaps on the decision boundaries get widened. As a result, different class clusters become farther away from each other.

D.4.1.2 Linear Interpolations using Regularized Autoencoders (related to Figure 6.5 in Section 6.4.1.1)

The generated point clouds from the representative intra-class linear interpolants between two cylinders and two cones with the regularized autoencoders under the Euclidean metric and info-Riemannian metric are drawn in Figure D.3.

D.4.2 Standard Benchmark Dataset

D.4.2.1 Performance Analysis with Varying Regularization Coefficients

Figure D.4 shows graphs of the classification accuracy versus reconstruction error for the trained AEs measured on ModelNet datasets, for a range of regularization coefficients. The reconstruction error is measured by the modified Chamfer distance as in [93]. Compared to vanilla autoencoders (red), regularized autoencoders under the info-Riemannian metric (blue) show overall higher classification accuracy regardless of the regularization coefficients. At the same time, they do not significantly increase the reconstruction error. On the other hand, when comparing the performance of regularized autoencoders under the Euclidean metric (green), most of these are clearly inferior to the vanilla autoencoder; the others perform even worse. Overall, regularization under the info-Riemannian metric is much more robust to the choice of regularization coefficients compared to using the Euclidean metric.

The linear SVM classification accuracy and reconstruction error (modified Chamfer distance) according to regularization coefficient are shown in Table D.4 and Table D.5. The tables are also arranged according to autoencoder models (FcNet vs. Foldingnet vs. PointCapsNet vs. DGCNN-FcNet) and regularization types (Vanilla vs. Euclidean vs. info-Riemannian).

D.4.2.2 Learning Curves for Noisy Point Cloud Data

Figure D.5 shows how the linear SVM classification accuracy and reconstruction error (modified Chamfer distance) evolve as the training proceeds, where datasets are ModelNet10 and ModelNet40 and noise levels are 1%, 5%, 10%, and 20% (details about noise are in Appendix D.3.4). Compared to vanilla autoencoders (light colored lines), regularized autoencoders under Info-Riemannian metrc (dark colored lines) show overall higher classification accuracy, while they show similar levels of reconstruction errors. The increase in classification accuracy becomes more pronounced as the noise level increases. Especially, when the noise level is 10% and 20%, the classification accuracy of the vanilla autoencoder and regularized autoencoder under Euclidean metric gradually decreases as the learning progresses (as the epoch increases). However, such phenomenons do not appear in the regularized autoencoders under the Info-Rimennian metric. This result implies that our method is very advantageous in situations where there is noise in the data.

D.4.2.3 Learning Curves for Semi-Supervised Classification

Regularized autoencoders (i.e., FcNet + I) show overall higher classification accuracy compared to vanilla autoencoders (i.e., FcNet), while their reconstruction errors are not significantly different to vanilla autoencoders'. Moreover, the increase in classification accuracy becomes more pronounced as the label rate decreases. In other words, our performance is more effective as the number of labels decreases. Figure D.6 shows how the linear SVM classification accuracy and reconstruction error evolve as the training proceeds, where label rate levels are 1%, 5%, 10%, and 50%. Also, similarly, when the label rate is 1%, the classification accuracy of the vanilla autoencoder gradually decreases as the learning progresses (as the epoch increases), but such phenomenons do not appear in the regularized autoencoders. This result implies that our method is also very advantageous in semi-supervised settings.

Table D.4: Classification accuracy and reconstruction error according to regularization coefficient. The table is also arranged according to model (FcNet vs. FoldingNet) and regularization type (Vanilla vs. Euclidean vs. info-Riemannian). For Riemannian metric cases, the regularization coefficients used in the actual experiments are σ^2 times λ shown in the table.

MODEL	METRIC	λ	MD40 acc	MD40 recon	MD10 acc	MD10 recon
FcNet	Euclidean	0.0001	89.343598	0.029069	92.951542	0.029527
FcNet	Euclidean	0.0010	88.330632	0.030464	93.612335	0.030684
FcNet	Euclidean	0.0100	88.249595	0.029877	93.722467	0.030054
FcNet	Euclidean	0.1000	86.993517	0.029878	92.841410	0.030900
FcNet	Euclidean	1.0000	87.115073	0.031966	92.951542	0.034272
FcNet	Euclidean	10.0000	86.709887	0.031523	92.400881	0.032318
FcNet	Euclidean	100.0000	85.696921	0.031841	92.511013	0.035172
FcNet	Euclidean	1000.0000	85.899514	0.035089	92.400881	0.036145
FcNet	Euclidean	10000.0000	86.345219	0.034149	92.400881	0.034811
FcNet	Riemannian	100.0000	89.586710	0.029032	94.052863	0.029435
FcNet	Riemannian	500.0000	89.829822	0.028944	94.273128	0.029928
FcNet	Riemannian	1000.0000	89.951378	0.028864	93.722467	0.029430
FcNet	Riemannian	2000.0000	90.194489	0.029165	94.052863	0.029606
FcNet	Riemannian	8000.0000	90.397083	0.028869	93.722467	0.028944
FcNet	Riemannian	10000.0000	89.991896	0.029603	94.162996	0.030278
FcNet	Riemannian	20000.0000	89.748784	0.028980	93.832599	0.029412
FcNet	Riemannian	50000.0000	89.667747	0.029161	93.392070	0.028894
FcNet	Riemannian	100000.0000	89.748784	0.028961	93.942731	0.030085
FcNet	Vanilla	0.0000	88.330632	0.028938	93.502203	0.029895
FoldingNet	Euclidean	0.0001	88.897893	0.030540	94.162996	0.029130
FoldingNet	Euclidean	0.0010	87.844408	0.031623	94.273128	0.031726
FoldingNet	Euclidean	0.0100	87.520259	0.029126	93.612335	0.032219
FoldingNet	Euclidean	0.1000	87.641815	0.029391	93.722467	0.030318
FoldingNet	Euclidean	1.0000	88.128039	0.030564	93.171806	0.031435
FoldingNet	Euclidean	10.0000	88.290113	0.032129	93.171806	0.032975
FoldingNet	Euclidean	100.0000	88.087520	0.032728	94.383260	0.032885
FoldingNet	Euclidean	1000.0000	87.722853	0.033305	92.951542	0.036244
FoldingNet	Euclidean	10000.0000	87.844408	0.033546	93.502203	0.036820
FoldingNet	Riemannian	100.0000	89.667747	0.029467	94.052863	0.030789
FoldingNet	Riemannian	500.0000	90.113452	0.029916	94.052863	0.029346
FoldingNet	Riemannian	1000.0000	89.870340	0.030090	94.493392	0.029996
FoldingNet	Riemannian	2000.0000	89.546191	0.030471	94.273128	0.031142
FoldingNet	Riemannian	8000.0000	89.627229	0.028949	94.162996	0.029745
FoldingNet	Riemannian	10000.0000	89.505673	0.029926	94.052863	0.033548
FoldingNet	Riemannian	20000.0000	89.384117	0.032798	94.162996	0.029463
FoldingNet	Riemannian	50000.0000	89.708266	0.029262	94.273128	0.030002
FoldingNet	Riemannian	100000.0000	89.262561	0.029511	94.162996	0.030355
FoldingNet	Vanilla	0.0000	89.343598	0.029599	93.722467	0.030528

Table D.5: Classification accuracy and reconstruction error according to regularization coefficient. The table is also arranged according to model (PointCapsNet vs. DGCNN-FcNet) and regularization type (Vanilla vs. Euclidean vs. info-Riemannian). For Riemannian metric cases, the regularization coefficients used in the actual experiments are σ^2 times λ shown in the table.

MODEL	METRIC	λ	MD40 acc	MD40 recon	MD10 acc	MD10 recon
PointCapsNet	Euclidean	0.0001	88.087520	0.039522	93.722467	0.043112
PointCapsNet	Euclidean	0.0010	87.601297	0.046227	93.171806	0.047749
PointCapsNet	Euclidean	0.0100	87.155592	0.058326	92.621145	0.060827
PointCapsNet	Euclidean	0.1000	86.628849	0.070735	91.519824	0.072615
PointCapsNet	Euclidean	1.0000	85.858995	0.087722	91.299559	0.087930
PointCapsNet	Euclidean	10.0000	83.954619	0.110708	90.638767	0.107402
PointCapsNet	Euclidean	100.0000	79.659643	0.124266	88.215859	0.115549
PointCapsNet	Euclidean	1000.0000	76.823339	0.128694	88.546256	0.121284
PointCapsNet	Euclidean	10000.0000	75.567261	0.130566	88.325991	0.121398
PointCapsNet	Riemannian	100.0000	88.492707	0.035034	93.942731	0.042224
PointCapsNet	Riemannian	1000.0000	88.168558	0.035816	93.942731	0.039131
PointCapsNet	Riemannian	2000.0000	87.884927	0.036824	93.392070	0.040169
PointCapsNet	Riemannian	5000.0000	87.884927	0.035990	93.832599	0.039729
PointCapsNet	Riemannian	8000.0000	87.641815	0.037986	93.392070	0.040947
PointCapsNet	Riemannian	10000.0000	87.763371	0.037507	93.722467	0.040428
PointCapsNet	Riemannian	20000.0000	87.763371	0.038454	93.281938	0.048909
PointCapsNet	Riemannian	50000.0000	87.641815	0.042233	93.281938	0.044333
PointCapsNet	Riemannian	100000.0000	87.317666	0.046607	93.171806	0.051187
PointCapsNet	Vanilla	0.0000	87.155592	0.033936	93.612335	0.037172
DGCNN-FcNet	Euclidean	0.0001	89.910859	0.028852	94.052863	0.029049
DGCNN-FcNet	Euclidean	0.0010	89.546191	0.029489	94.383260	0.029480
DGCNN-FcNet	Euclidean	0.0100	88.492707	0.030448	93.061674	0.030531
DGCNN-FcNet	Euclidean	0.1000	87.884927	0.030046	93.392070	0.030966
DGCNN-FcNet	Euclidean	1.0000	87.520259	0.030605	93.502203	0.034671
DGCNN-FcNet	Euclidean	10.0000	87.196110	0.032174	92.841410	0.032151
DGCNN-FcNet	Euclidean	100.0000	87.277147	0.032385	93.171806	0.033750
DGCNN-FcNet	Euclidean	1000.0000	87.682334	0.033359	93.171806	0.033906
DGCNN-FcNet	Euclidean	10000.0000	86.385737	0.034471	92.621145	0.036056
DGCNN-FcNet	Riemannian	100.0000	90.397083	0.028761	94.493392	0.028591
DGCNN-FcNet	Riemannian	1000.0000	90.964344	0.028278	94.493392	0.028806
DGCNN-FcNet	Riemannian	2000.0000	90.680713	0.028299	94.493392	0.028832
DGCNN-FcNet	Riemannian	5000.0000	90.883306	0.028520	94.493392	0.029128
DGCNN-FcNet	Riemannian	8000.0000	90.802269	0.028727	95.154185	0.028971
DGCNN-FcNet	Riemannian	10000.0000	90.680713	0.028583	94.383260	0.028820
DGCNN-FcNet	Riemannian	20000.0000	90.559157	0.028677	94.713656	0.029067
DGCNN-FcNet	Riemannian	50000.0000	90.761750	0.029794	94.713656	0.029695
DGCNN-FcNet	Riemannian	100000.0000	90.235008	0.028934	94.052863	0.031275
DGCNN-FcNet	Vanilla	0.0000	90.275527	0.028867	94.493392	0.029454

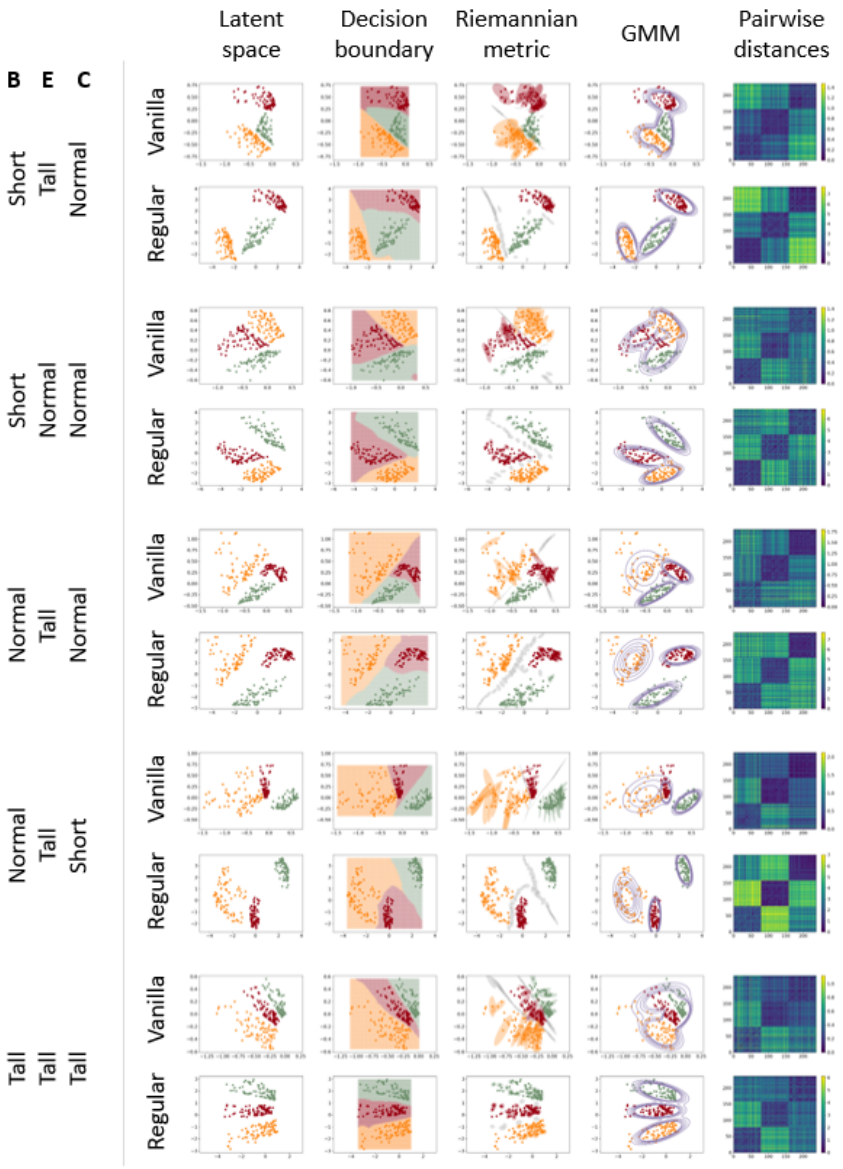


Figure D.2: The representative five examples of the regularization experiments on the synthetic dataset. From left to right: latent spaces, decision boundary according to the color assigning method introduced in Appendix D.3.2, latent spaces with equidistant ellipse ($\{z|(z-z^*)^T G(z^*)(z-z^*) = 1\}$ for center z^*) centered on some selected points and sampled points from interspaces, Gaussian Mixture Model (GMM) fitting results, and the heat map of the pairwise Euclidean distances in the latent space of all test data. For each experiment, the upper figure is a vanilla autoencoder trained without regularization, while the lower figure is trained with regularization.

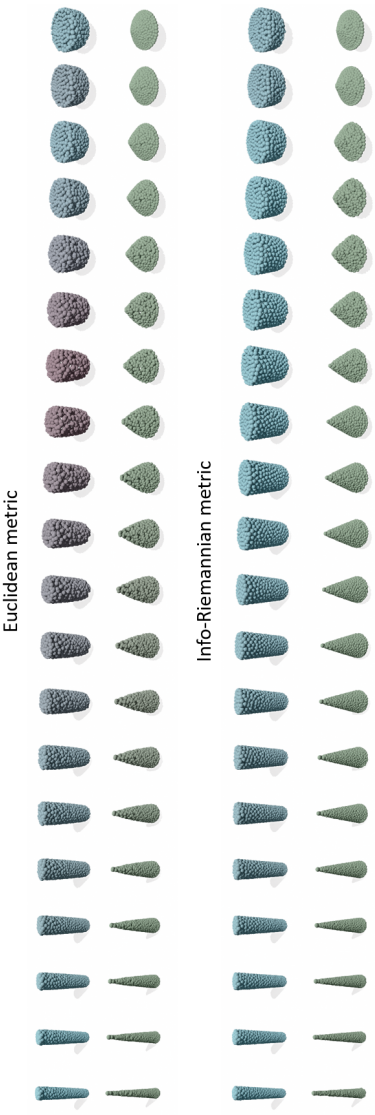


Figure D.3: The generated point clouds from the linear interpolants of the regularized autoencoders with the Euclidean metric (*Upper*) and info-Riemannian metric (*Lower*).

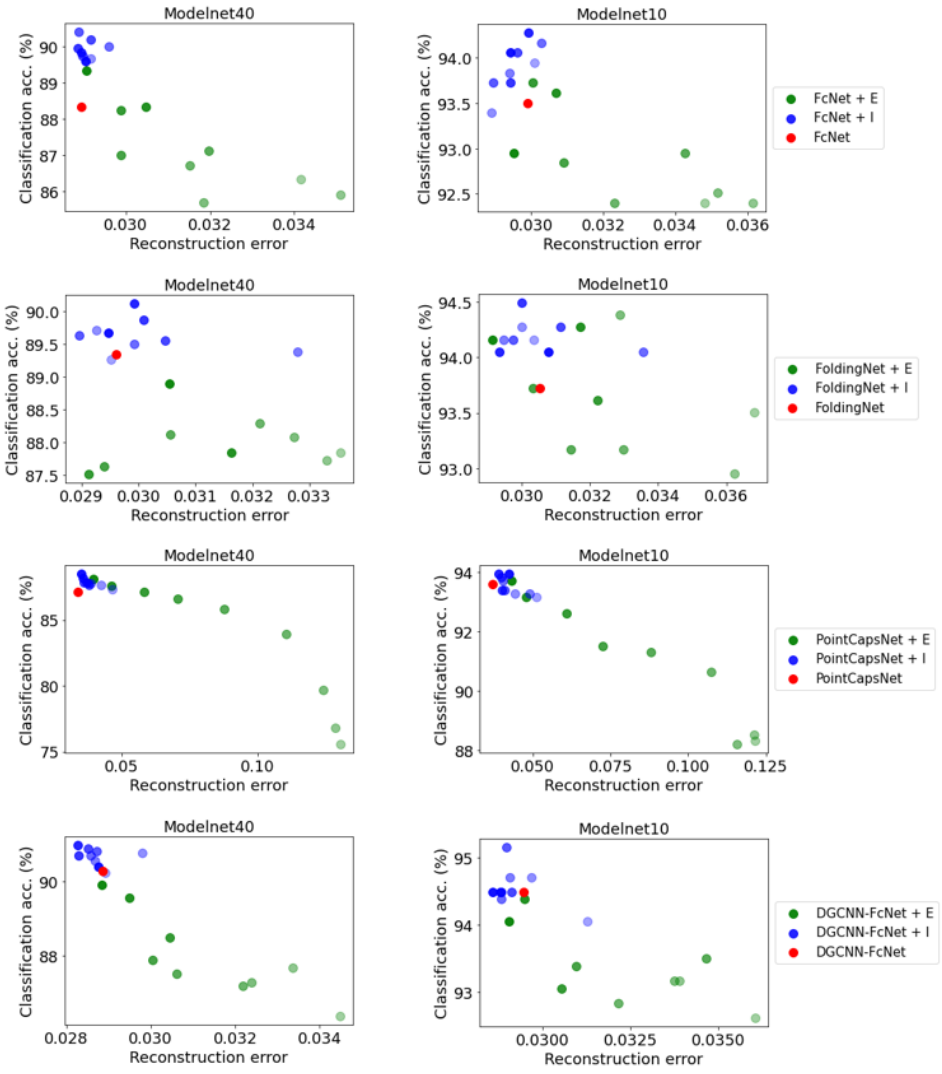


Figure D.4: Graphs of classification accuracy versus reconstruction error measured on ModelNet datasets. More transparent markers have larger coefficients λ; detailed values are in Table D.4 and Table D.5.

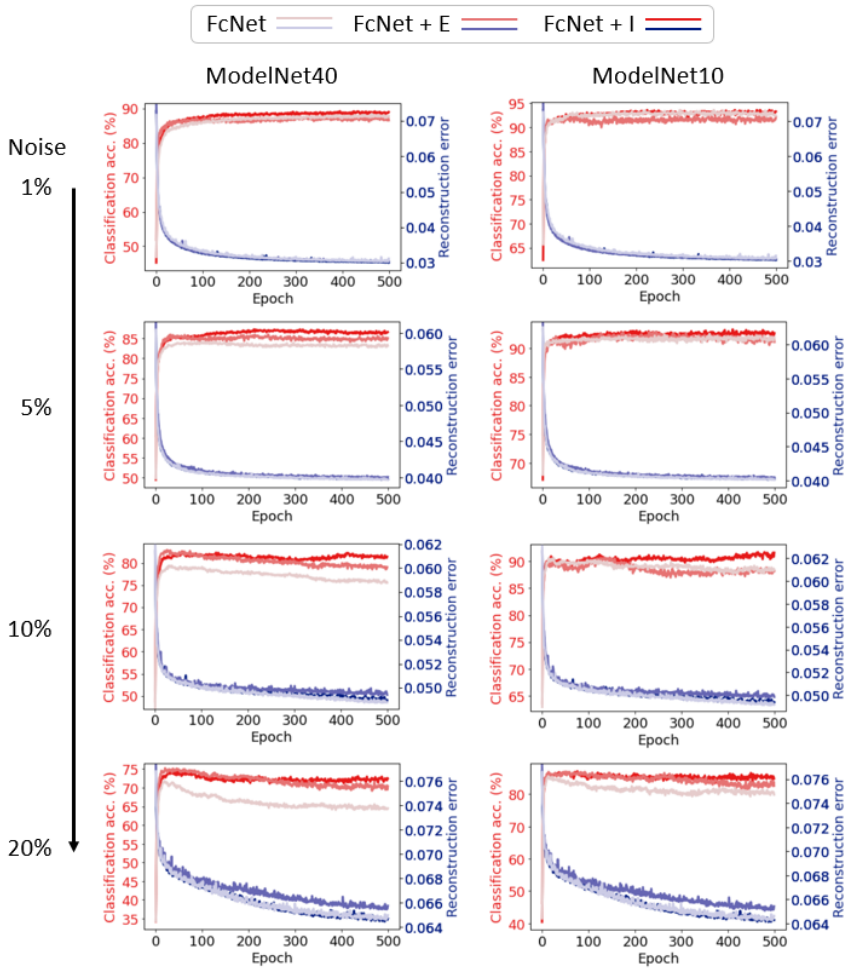


Figure D.5: Learning curves of classification accuracy and reconstruction error measured on ModelNet datasets (ModelNet40 and ModelNet10) according to the noise levels (1%, 5%, 10%, and 20%). In each plot, the light colored lines are the result of the non-regularized autoencoders (i.e., FcNet), and the dark colored lines are the result of the regularized autoencoders (i.e., FcNet + E and FcNet + I).

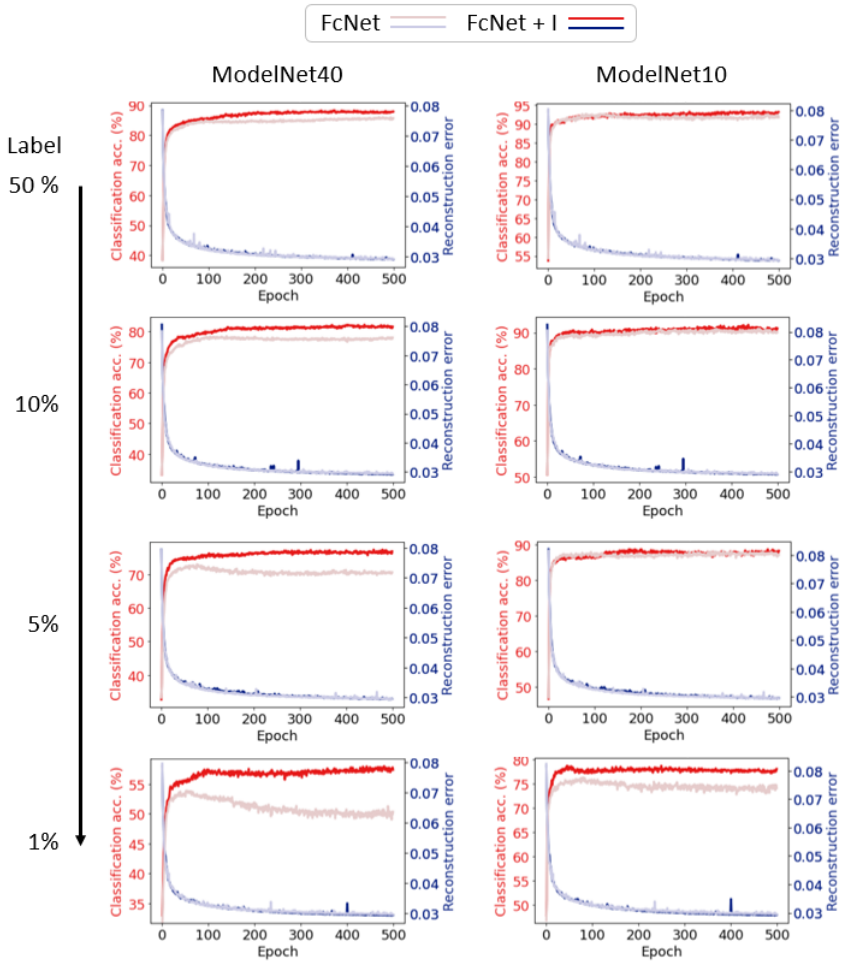


Figure D.6: Learning curves of classification accuracy and reconstruction error measured on ModelNet datasets (ModelNet40 and ModelNet10) according to the label rates (50%, 10%, 5%, and 1%). In each plot, the light colored lines are the result of the non-regularized autoencoders (i.e., FcNet), and the dark colored lines are the result of the regularized autoencoders (i.e., FcNet + I).

Bibliography

[1] Gerard V Trunk. A problem of dimensionality: A simple example. *IEEE Transactions on pattern analysis and machine intelligence*, (3):306–307, 1979.

[2] Lawrence Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005.

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

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

[5] 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.

[6] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

[7] Zhenyue Zhang and Hongyuan Zha. Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *SIAM journal on scientific computing*, 26(1):313–338, 2004.

[8] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[9] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[10] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[12] Cheongjae Jang. *Riemannian Distortion Measures for Non-Euclidean Data*. PhD thesis, Seoul National University Graduate School, 2019.

[13] Yonghyeon Lee, Hyeokjun Kwon, and Frank Park. Neighborhood reconstructing autoencoders. *Advances in Neural Information Processing Systems*, 34:536–546, 2021.

[14] Yonghyeon Lee, Sangwoong Yoon, MinJun Son, and Frank C. Park. Regularized autoencoders for isometric representation learning. In *International Conference on Learning Representations*, 2022.

[15] Yonghyeon Lee, Seungyeon Kim, Jinwon Choi, and Frank Park. A statistical manifold framework for point cloud data. In *International Conference on Machine Learning*, pages 12378–12402. PMLR, 2022.

[16] Minyeon Han and Frank C Park. Dti segmentation and fiber tracking using metrics on multivariate normal distributions. *Journal of mathematical imaging and vision*, 49(2):317–334, 2014.

[17] Georgios Arvanitidis, Lars Kai Hansen, and Søren Hauberg. Latent space oddity: on the curvature of deep generative models. *arXiv preprint arXiv:1710.11379*, 2017.

[18] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[19] Michael Moor, Max Horn, Bastian Rieck, and Karsten Borgwardt. Topological autoencoders. In *International Conference on Machine Learning*, pages 7045–7054. PMLR, 2020.

[20] Simon Schönenberger, Anastasiia Varava, Vladislav Polianskii, Jen Jen Chung, Roland Siegwart, and Danica Kragic. Witness autoencoder: Shaping the latent space with witness complexes.

[21] Andrés F Duque, Sacha Morin, Guy Wolf, and Kevin R Moon. Extendable and invertible manifold learning with geometry regularized autoencoders. *arXiv preprint arXiv:2007.07142*, 2020.

[22] Xingyu Chen, Chunyu Wang, Xuguang Lan, Nanning Zheng, and Wenjun Zeng. Neighborhood geometric structure-preserving variational autoencoder for smooth and bounded data sources. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[23] Mukund Balasubramanian, Eric L Schwartz, Joshua B Tenenbaum, Vin de Silva, and John C Langford. The isomap algorithm and topological stability. *Science*, 295(5552):7–7, 2002.

[24] CJ Carey. Graph construction for manifold discovery. 2017.

[25] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[26] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[27] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

[28] Nat Dilokthanakul, Pedro AM Mediano, Marta Garnelo, Matthew CH Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

[29] Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

[30] Jakub Tomczak and Max Welling. Vae with a vampprior. In *International Conference on Artificial Intelligence and Statistics*, pages 1214–1223. PMLR, 2018.

[31] Alexej Klushyn, Nutan Chen, Richard Kurle, Botond Cseke, and Patrick van der Smagt. Learning hierarchical priors in vaes. *arXiv preprint arXiv:1905.04982*, 2019.

[32] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Icml*, 2011.

[33] Nutan Chen, Alexej Klushyn, Francesco Ferroni, Justin Bayer, and Patrick Van Der Smagt. Learning flat latent manifolds with vaes. *arXiv preprint arXiv:2002.04881*, 2020.

[34] Wei Wang, Yan Huang, Yizhou Wang, and Liang Wang. Generalized autoencoder: A neural network framework for dimensionality reduction. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 490–497, 2014.

[35] Manfredo P Do Carmo. *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016.

[36] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *arXiv preprint arXiv:1706.08500*, 2017.

[37] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

[38] Vanessa Böhm and Uroš Seljak. Probabilistic auto-encoder. *arXiv preprint arXiv:2006.05479*, 2020.

[39] Sangwoong Yoon, Yung-Kyun Noh, and Frank Chongwoo Park. Autoencoding under normalization constraints. *arXiv preprint arXiv:2105.05735*, 2021.

[40] I Tolstikhin, O Bousquet, S Gelly, and B Schölkopf. Wasserstein auto-encoders. In *6th International Conference on Learning Representations (ICLR 2018)*. OpenReview. net, 2018.

[41] Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593, 2014.

[42] Thomas Bendokat, Ralf Zimmermann, and P-A Absil. A grassmann manifold handbook: Basic geometry and computational aspects. *arXiv preprint arXiv:2011.13699*, 2020.

[43] James Eells and Luc Lemaire. A report on harmonic maps. *Bulletin of the London mathematical society*, 10(1):1–68, 1978.

[44] James Eells and Luc Lemaire. Another report on harmonic maps. *Bulletin of the London Mathematical Society*, 20(5):385–524, 1988.

[45] Frank C Park and Roger W Brockett. Kinematic dexterity of robotic mechanisms. *The International Journal of Robotics Research*, 13(1):1–15, 1994.

[46] Cheongjae Jang, Yung-Kyun Noh, and Frank Chongwoo Park. A riemannian geometric framework for manifold learning of non-euclidean data. *Advances in Data Analysis and Classification*, pages 1–27, 2020.

[47] Wolfgang Kühnel. *Differential geometry*, volume 77. American Mathematical Soc., 2015.

[48] Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

[49] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *Proceedings of the International Conference on Learning Representations*, 2019.

[50] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1010–1019, 2016.

[51] Dhananjay Bhaskar, Kincaid MacDonald, Dawson Thomas, Sarah Zhao, Kisung You, Jennifer Paige, Yariv Aizenbud, Bastian Rieck, Ian M Adelstein, and Smita Krishnaswamy. Diffusion-based methods for estimating curvature in data. In *ICLR 2022 Workshop on Geometrical and Topological Representation Learning*, 2022.

[52] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning. *arXiv preprint arXiv:1812.05069*, 2018.

[53] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[54] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Information maximizing variational autoencoders. *arXiv preprint arXiv:1706.02262*, 2017.

[55] Michael S Lewicki and Terrence J Sejnowski. Learning overcomplete representations. *Neural computation*, 12(2):337–365, 2000.

[56] Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. Pixelvae: A latent variable model for natural images. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[57] James McQueen, Marina Meila, and Dominique Perrault-Joncas. Nearly isometric embedding by relaxation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 2639–2647, 2016.

[58] Erez Peterfreund, Ofir Lindenbaum, Felix Dietrich, Tom Bertalan, Matan Gavish, Ioannis G Kevrekidis, and Ronald R Coifman. Local conformal autoencoder for standardized data coordinates. *Proceedings of the National Academy of Sciences*, 117(49):30918–30927, 2020.

[59] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[60] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[61] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A riemannian framework for tensor computing. *International Journal of computer vision*, 66(1):41–66, 2006.

[62] Taeyoon Lee and Frank C Park. A geometric algorithm for robust multibody inertial parameter identification. *IEEE Robotics and Automation Letters*, 3(3):2455–2462, 2018.

[63] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[65] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.

[66] Felix Hausdorff. Grundzüge der mengenlehre. 1914.

[67] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[68] Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191. American Mathematical Soc., 2000.

[69] Shun-ichi Amari. *Information geometry and its applications*, volume 194. Springer, 2016.

[70] Bing Jian and Baba C Vemuri. A robust algorithm for point set registration using mixture of gaussians. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1246–1251. IEEE, 2005.

[71] Fei Wang, Baba C Vemuri, and Anand Rangarajan. Groupwise point pattern registration using a novel cdf-based jensen-shannon divergence. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 1283–1288. IEEE, 2006.

[72] Andriy Myronenko and Xubo Song. Point set registration: Coherent point drift. *IEEE transactions on pattern analysis and machine intelligence*, 32(12):2262–2275, 2010.

[73] Erion Hasanbelliu, Luis Sanchez Giraldo, and José C Príncipe. Information theoretic shape matching. *IEEE transactions on pattern analysis and machine intelligence*, 36(12):2436–2451, 2014.

[74] Zhiyong Zhou, Jian Zheng, Yakang Dai, Zhe Zhou, and Shi Chen. Robust non-rigid point set registration using student's-t mixture model. *PloS one*, 9(3):e91381, 2014.

[75] Zhe Min, Jiaole Wang, and Max Q-H Meng. Robust generalized point cloud registration using hybrid mixture model. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4812–4818. IEEE, 2018.

[76] Feiran Li, Kent Fujiwara, and Yasuyuki Matsushita. Toward a unified framework for point set registration. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12981–12987. IEEE, 2021.

[77] Faisal Zaman, Ya Ping Wong, and Boon Yian Ng. Density-based denoising of point cloud. In *9th International Conference on Robotic, Vision, Signal Processing and Power Applications*, pages 287–295. Springer, 2017.

[78] Shitong Luo and Wei Hu. Score-based point cloud denoising. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4583–4592, 2021.

[79] Yonghyeon Lee, Sangwoong Yoon, Minjun Son, and Frank Park. Regularized autoencoders for isometric representation learning. *International Conference on Learning Representation*, 2022.

[80] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[81] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[82] Bradley Efron and David V Hinkley. Assessing the accuracy of the maximum likelihood estimator: Observed versus expected fisher information. *Biometrika*, 65(3):457–483, 1978.

[83] Jorma J Rissanen. Fisher information and stochastic complexity. *IEEE transactions on information theory*, 42(1):40–47, 1996.

[84] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[85] Richard A Davis, Keh-Shin Lii, and Dimitris N Politis. Remarks on some nonparametric estimates of a density function. In *Selected Works of Murray Rosenblatt*, pages 95–100. Springer, 2011.

[86] Ben Knudsen. Configuration spaces in algebraic topology. *arXiv preprint arXiv:1803.11165*, 2018.

[87] Bharath K Sriperumbudur, Arthur Gretton, Kenji Fukumizu, Bernhard Schölkopf, and Gert RG Lanckriet. Hilbert space embeddings and metrics on probability measures. *The Journal of Machine Learning Research*, 11:1517–1561, 2010.

[88] Hang Shao, Abhishek Kumar, and P Thomas Fletcher. The riemannian geometry of deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 315–323, 2018.

[89] Tao Yang, Georgios Arvanitidis, Dongmei Fu, Xiaogang Li, and Søren Hauberg. Geodesic clustering in deep generative models. *arXiv preprint arXiv:1809.04747*, 2018.

[90] Nutan Chen, Alexej Klushyn, Richard Kurle, Xueyan Jiang, Justin Bayer, and Patrick Smagt. Metrics for deep generative models. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1550. PMLR, 2018.

[91] Dimitris Kalatzis, David Eklund, Georgios Arvanitidis, and Søren Hauberg. Variational autoencoders with riemannian brownian motion priors. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

[92] Georgios Arvanitidis, Søren Hauberg, and Bernhard Schölkopf. Geometrically enriched latent spaces. *arXiv preprint arXiv:2008.00565*, 2020.

[93] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 206–215, 2018.

[94] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018.

[95] Theo Deprelle, Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. Learning elementary structures for 3d shape generation and matching. *arXiv preprint arXiv:1908.04725*, 2019.

[96] Jiahao Pang, Duanshun Li, and Dong Tian. Tearingnet: Point cloud autoencoder to learn topology-friendly representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7453–7462, 2021.

[97] Cem Yuksel. Sample elimination for generating poisson disk sample sets. In *Computer Graphics Forum*, volume 34, pages 25–32. Wiley Online Library, 2015.

[98] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[99] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3d point capsule networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1009–1018, 2019.

[100] Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Rotation invariant spherical harmonic representation of 3 d shape descriptors. In *Symposium on geometry processing*, volume 6, pages 156–164, 2003.

[101] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*, volume 22, pages 223–232. Wiley Online Library, 2003.

[102] Abhishek Sharma, Oliver Grau, and Mario Fritz. Vconv-dae: Deep volumetric shape learning without object labels. In *European Conference on Computer Vision*, pages 236–250. Springer, 2016.

[103] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 82–90, 2016.

[104] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pages 40–49. PMLR, 2018.

[105] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4541–4550, 2019.

[106] Kaveh Hassani and Mike Haley. Unsupervised multi-task feature learning on point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8160–8171, 2019.

[107] Yung-Kyun Noh, Byoung-Tak Zhang, and Daniel Lee. Generative local metric learning for nearest neighbor classification. *Advances in Neural Information Processing Systems*, 23, 2010.

[108] Jun Wang, Alexandros Kalousis, and Adam Woznica. Parametric local metric learning for nearest neighbor classification. *Advances in neural information processing systems*, 25, 2012.

[109] Søren Hauberg, Oren Freifeld, and Michael Black. A geometric take on metric learning. *Advances in Neural Information Processing Systems*, 25, 2012.

[110] Yung-Kyun Noh, Masashi Sugiyama, Kee-Eung Kim, Frank Park, and Daniel D Lee. Generative local metric learning for kernel regression. *Advances in Neural Information Processing Systems*, 30, 2017.

[111] Tim R Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M Tomczak. Hyperspherical variational auto-encoders. *arXiv preprint arXiv:1804.00891*, 2018.

[112] Luca Falorsi, Pim De Haan, Tim R Davidson, Nicola De Cao, Maurice Weiler, Patrick Forré, and Taco S Cohen. Explorations in homeomorphic variational auto-encoding. *arXiv preprint arXiv:1807.04689*, 2018.

[113] Bradley CA Brown, Anthony L Caterini, Brendan Leigh Ross, Jesse C Cresswell, and Gabriel Loaiza-Ganem. The union of manifolds hypothesis and its implications for deep generative modelling. *arXiv preprint arXiv:2207.02862*, 2022.

[114] Stefan Schonsheck, Jie Chen, and Rongjie Lai. Chart auto-encoders for manifold structured data. *arXiv preprint arXiv:1912.10094*, 2019.

[115] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[116] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. *arXiv preprint arXiv:1711.00848*, 2017.

[117] Ricky TQ Chen, Xuechen Li, Roger Grosse, and David Duvenaud. Isolating sources of disentanglement in variational autoencoders. *arXiv preprint arXiv:1802.04942*, 2018.

[118] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In *International Conference on Machine Learning*, pages 2649–2658. PMLR, 2018.

[119] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pages 4114–4124. PMLR, 2019.

[120] Ilyes Khemakhem, Diederik Kingma, Ricardo Monti, and Aapo Hyvarinen. Variational autoencoders and nonlinear ica: A unifying framework. In *International Conference on Artificial Intelligence and Statistics*, pages 2207–2217. PMLR, 2020.

[121] Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018.

[122] Marco Fumero, Luca Cosmo, Simone Melzi, and Emanuele Rodolà. Learning disentangled representations via product manifold projection. *arXiv preprint arXiv:2103.01638*, 2021.

[123] Hugo Caselles-Dupré, Michael Garcia Ortiz, and David Filliat. Symmetry-based disentangled representation learning requires interaction with environments. *Advances in Neural Information Processing Systems*, 32:4606–4615, 2019.

[124] Robin Quessard, Thomas D Barrett, and William R Clements. Learning group structure and disentangled representations of dynamical environments. *arXiv preprint arXiv:2002.06991*, 2020.

[125] Daniella Horan, Eitan Richardson, and Yair Weiss. When is unsupervised disentanglement possible? *Advances in Neural Information Processing Systems*, 34:5150–5161, 2021.

[126] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[127] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

[128] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[129] William S Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368):829–836, 1979.

[130] Salah Rifai, Grégoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher order contractive auto-encoder. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 645–660. Springer, 2011.

[131] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 2180–2188, 2016.

[132] Elizaveta Levina and Peter J Bickel. Maximum likelihood estimation of intrinsic dimension. In *Advances in neural information processing systems*, pages 777–784, 2005.

[133] Felix Hausdorff. *Felix Hausdorff-Gesammelte Werke Band III: Mengenlehre (1927, 1935) Deskripte Mengenlehre und Topologie*, volume 3. Springer-Verlag, 2008.

[134] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.

[135] Trung Nguyen, Quang-Hieu Pham, Tam Le, Tung Pham, Nhat Ho, and Binh-Son Hua. Point-set distances for learning representations of 3d point clouds. *arXiv preprint arXiv:2102.04014*, 2021.

[136] Luca Cosmo, Antonio Norelli, Oshri Halimi, Ron Kimmel, and Emanuele Rodolà. Limp: Learning latent shape representations with metric preservation priors. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 19–35. Springer, 2020.

[137] Vern I Paulsen and Mrinal Raghupathi. *An introduction to the theory of reproducing kernel Hilbert spaces*, volume 152. Cambridge university press, 2016.

[138] Yonghyeon Lee, Jonghyuk Baek, Young Min Kim, and Frank Chongwoo Park. Imat: The iterative medial axis transform. In *Computer Graphics Forum*. Wiley Online Library, 2021.

국문초록

실세계 문제에서 우리가 관측하는 데이터들은 대부분 고차원이며, 이러한 고차원 데이터에 기계학습 알고리즘을 바로 적용하기는 매우 어렵다. 다양체 가정에서는 고차원의 데이터가 실제로는 더 작은 차원의 다양체 위에 놓여 있다고 가정한다. 즉, 초기에 데이터를 표현하는 변수의 수보다 훨씬 적은 수의 변수로 데이터의 표현이 가능하다는 것이다. 데이터가 놓여있는 저차원의 다양체 구조와 그것의 표현을 알아내는 것, 즉 다양체 표현 학습은 기계 학습에서 중요한 문제 중 하나이다.

오토인코더는 고차원의 데이터를 저차원의 표현 공간으로 보내주는 인코더와, 저차원의 인코딩된 값을 다시 고차원의 데이터 공간으로 보내주는 디코더로 이루어져 있으며, 저차원의 다양체 구조와 표현을 학습하는 데 널리 사용되고 있다. 본 학위논문에서는, 이러한 오토인코더의 두 가지 문제점을 발견하고 해결한다. 첫 번째 문제점은 오토인코더가 노이즈에 과적합 된 다양체를 학습하거나 잘못된 기하학적 연결 관계를 가지는 다양체를 학습하는 것이고, 둘째는 오토인코더가 데이터 간의 거리와 각도 관계를 보존하지 못하는 기하학적으로 왜곡된 표현공간을 학습하는 문제이다.

학습하고자 하는 다양체는 보통 곡률이 있기 때문에, 기저의 기하학적 특성을 고려하여 좌표계에 의존하지 않는 알고리즘을 설계하는 것이 매우 중요하다. 기존의 오토인코더 방법들은 대부분 인코더에 의해 완전히 결정되는 표현공간에 집중했으며, 디코더에는 거의 관심을 가지지 않았고, 특히 데이터의 기저에 있는 기하학적인 특성을 반영하고 있지 않다. 본 학위논문은 이러한 단점들을 해결하는 새로운 오토인코더 기반 다양체 학습 알고리즘들을 제안한다. 본 논문에서의 흥미로운 발견 중의 하나는, 오토인코더 기반 다양체 학습에서, 디코더가 인코더와 동등하게 혹은 때때로 더 중요한 역할을 한다는 점이다.

제안된 기하학적 방법들은 사전에 만들어진 이웃 그래프 혹은 리마니안 메트릭을 활용하여 정확한 다양체와 기하를 보존하는 표현을 학습하기 위한 새로운 손실 함수들을 제안하였다. 특히, 리만 기하학을 활용한 방법에서는, 정규화 항이 좌표계 불변성을 만족하도록 설계하여 기하학적으로 유의미한 값을 측정하도록 하였다. 다양한 사진, 모션 캡처, 점 구름 데이터를 이용한 실험을 통해, 기존에 존재하는 최신의 오토인코더 방법 대비, 우리의 방법이 다양체를 더 정확하게 더 적은 왜곡으로 학습함을 보였다.

또한, 사진 검색, 군집화, 분류 등의 다양한 다운스트림 작업의 성능이 향상됨을 보였다.

주요어: 다양체 학습, 오토인코더, 리만 기하학, 아이소메트릭 표현

학번: 2018-20161