Graph Geometry-Preserving Autoencoders

Jungbin Lim* [1] **Jihwan Kim*** [1] **Yonghyeon Lee** [2] **Cheongjae Jang** [3] **Frank Chongwoo Park** [1]

Abstract

When using an autoencoder to learn the low-dimensional manifold of high-dimensional data, it is crucial to find the latent representations that preserve the geometry of the data manifold. However, most existing studies assume a Euclidean nature for the high-dimensional data space, which is arbitrary and often does not precisely reflect the underlying semantic or domain-specific attributes of the data. In this paper, we propose a novel autoencoder regularization framework based on the premise that the geometry of the data manifold can often be better captured with a well-designed similarity graph associated with data points. Given such a graph, we utilize a Riemannian geometric distortion measure as a regularizer to preserve the geometry derived from the graph Laplacian and make it suitable for larger-scale autoencoder training. Through extensive experiments, we show that our method outperforms existing state-of-the-art geometry-preserving and graph-based autoencoders with respect to learning accurate latent structures that preserve the graph geometry, and is particularly effective in learning dynamics in the latent space. Code is available at https://github.com/JungbinLim/GGAE-public.

1. Introduction

Autoencoders are widely used to learn the low-dimensional manifold of high-dimensional data points and to find their latent representations (Arvanitidis et al., 2017; Shao et al., 2018; Lee & Park, 2023; Jang et al., 2022; Lee, 2023). It has been widely observed that vanilla autoencoders often fail to produce latent representations that preserve the geometry of the data manifold, i.e., actual distances and angles on the manifold do not match those measured in the latent space. These distortions can be effectively mitigated by applying appropriate regularization techniques, as pointed out and validated via a wide range of case studies in Chen et al. (2020); Lee et al. (2022b); Nazari et al. (2023).

Most autoencoder regularization techniques assume a Euclidean ambient data space and its standard geometry, and attempt to preserve distances, angles, or volumes on the data manifold (Chen et al., 2020; Lee et al., 2022b; Nazari et al., 2023; Singh & Nag, 2021). It is worth noting that for manifold learning purposes, the Euclidean ambient space assumption is arbitrary; other non-Euclidean ambient space metrics may be more appropriate for the problem at hand. While some studies have tried to estimate the ambient space metric (Alipanahi et al., 2008; Hauberg et al., 2012; Arvanitidis et al., 2020), these methods are for the most part quite computation- and memory-intensive. Some approaches like Arvanitidis et al. (2020) assume a diagonal ambient space metric, but such simplifying assumptions may not accurately reflect the data's underlying semantic or domain-specific attributes, failing to capture the correct geometry of the problem and leading to poor downstream task performance.

The essence of the data space can often be captured by using a well-designed similarity graph associated with data points, in which a node represents each data point, and an edge represents the similarity between a data pair (Costa & Hero, 2004; Belkin & Niyogi, 2004; Chong et al., 2020). Compared to obtaining explicit labels, these graphs provide valuable information for semantic or domain-specific properties of the data in a relatively cost-effective manner. We aim to leverage the geometry derived from these graphs in finding better representations obtained from autoencoders.

In this paper, we propose a novel autoencoder regularization framework, the *Graph Geometry-Preserving Autoencoder (GGAE)*, to find representations that attempt to preserve the geometry of the data manifold implied by a given similarity graph. We use an established Riemannian geometric distortion measure as a regularizer (Jang et al., 2021; Lee et al., 2022b), quantifying how a mapping between two Riemannian manifolds deviates from an isometry that preserves

*Equal contribution [1]Department of Mechanical Engineering, Seoul National University, Seoul, Republic of Korea [2]Center for AI and Natural Sciences, Korea Institute for Advanced Study, Seoul, Republic of Korea [3]The AI Institute, Hanyang University, Seoul, Republic of Korea. Correspondence to: Frank Chongwoo Park <fcp@snu.ac.kr>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

distances and angles.

In order to derive a metric from a graph without explicitly estimating the high-dimensional ambient metric, we expand the graph Laplacian-based estimation techniques employed in Rosenberg (1997); Hein et al. (2007); Belkin et al. (2009); Perraul-Joncas & Meila (2013); Jang et al. (2021) to the autoencoder training setting. One challenge in such extensions is that, since these techniques require latent coordinates of *all* data points, the entire dataset must be encoded repeatedly at each iteration of the gradient descent. To make this method suitable for larger scale autoencoder training, we consider two batch-based graph Laplacian approximation methods: (i) *Laplacian-Slicing*, and (ii) *Batch-Kernel*. As we show later via empirical studies, our studies indicate that the latter method is superior to the former (see Appendix A).

We perform extensive experiments comparing different types of geometry-preserving and graph-based autoencoders, and demonstrate that our method is the most effective in learning latent structures that preserve the geometry of the given graph across diverse datasets. In particular, when compared to SPAE (Singh & Nag, 2021), which is most aligned with our objective, our method shows enhanced robustness to errors in global geodesic distances within the graph and in sparse graph cases where some edges are missing. Through case studies involving high-dimensional images of a robot manipulator, we demonstrate that GGAE can provide suitable low-dimensional latent representations for learning the latent dynamics of a robot system.

2. Geometry Preserving Mapping and Distortion Measure

As preliminaries, this section first introduces a geometry-preserving mapping, an isometry, between Riemannian manifolds, followed by a distortion measure that measures the proximity of a given mapping to an isometry. Lastly, we introduce a technique to estimate the distortion measure from finite samples on Riemannian manifolds. For relevant references, we refer to Eells & Sampson (1964); Coifman & Lafon (2006); Jang et al. (2021); Lee et al. (2022b).

2.1. Geometry-Preserving Mapping between Riemannian Manifolds

Let $\mathcal{M}$ be a Riemannian manifold of dimension m with local coordinates $x \in \mathbb{R}^m$ and Riemannian metric $G(x) \in \mathbb{R}^{m\times m}$, and $\mathcal{N}$ be a Riemannian manifold of dimension n with local coordinates $z \in \mathbb{R}^n$ and Riemannian metric $H(z) \in \mathbb{R}^{n\times n}$. Let $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ be a smooth mapping, represented in local coordinates by $f : \mathbb{R}^m \to \mathbb{R}^n$. The differential at each $\mathrm{x} \in \mathcal{M}$ can be represented in local coordinates by the Jacobian matrix $J_f(x) := \frac{\partial f}{\partial x} \in \mathbb{R}^{n\times m}$.

A mapping is called an *isometry* if it preserves distances, angles, and volumes everywhere. In the case of a mapping $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ between Riemannian manifolds, f is a local isometry at $\mathrm{x} \in \mathcal{M}$, if

$$G(x) = J_f(x)^\top H(f(x)) J_f(x). \tag{1}$$

If (1) holds for every x in $\mathcal{M}$, then f is called a global isometry. In this paper, we consider a positive measure μ in $\mathcal{M}$ and f is considered an isometry if (1) holds for all x in the support of μ.

2.2. Distortion Measure of Isometry

Considering (1), the distortion introduced by $\mathrm{f} : \mathcal{M} \to \mathcal{N}$ at each point $\mathrm{x} \in \mathcal{M}$ can be measured as a difference between $J_f(x)^\top H(f(x)) J_f(x)$ and $G(x)$. This difference can be measured in a coordinate-invariant way using the eigenvalues of $J_f^\top H J_f G^{-1} \in \mathbb{R}^{m\times m}$ [1] (Jang et al., 2021; Lee et al., 2022b). For (1) to hold, the eigenvalues must all equal one. A simple choice to measure the deviation from this ideal case at each point $x \in \mathbb{R}^m$ is

$$\sum_{i=1}^{m} (\lambda_i(x) - 1)^2 = m + \sum_{i=1}^{m} \lambda_i(x)^2 - 2\lambda_i(x), \tag{2}$$

where $\lambda_i(x)$ denotes the i-th eigenvalue of the $m \times m$ matrix $J_f(x)^\top H(f(x)) J_f(x) G^{-1}(x)$.

To measure the amount of global distortion induced by f, we need some means to integrate the point-wise distortion measure (2) over $\mathcal{M}$. This can be done by integrating (2) with respect to some positive measure μ in $\mathcal{M}$. The corresponding global measure of distortion is

$$\int_{\mathcal{M}} \mathrm{Tr}((J_f^\top H J_f G^{-1})^2 - 2 J_f^\top H J_f G^{-1})\, d\mu, \tag{3}$$

where we ignore the constant term in (2) and use that the trace of a matrix equals the sum of its eigenvalues. Lastly, we note that since the trace is invariant to the cyclic permutation of matrices, we have an equivalent form of (3):

$$\int_{\mathcal{M}} \mathrm{Tr}((H J_f G^{-1} J_f^\top)^2 - 2 H J_f G^{-1} J_f^\top)\, d\mu. \tag{4}$$

2.3. Estimation of $J_f G^{-1} J_f^\top$

Suppose we are given only a finite set of points $\mathrm{x}_i \in \mathcal{M}$, $i = 1, \ldots, N$, neither local coordinates for $\mathcal{M}$ nor a coordinate representation of its Riemannian metric. Adopting ideas from Perraul-Joncas & Meila (2013), we can approximate the distortion measure (4) from the finite samples, if we are given (i) local coordinates $z \in \mathbb{R}^n$ and metric $H(z)$ on the

[1] Apart from their order, the eigenvalues are invariant under coordinate transformations $x \mapsto \psi(x)$, $z \mapsto \xi(z)$.

target manifold $\mathcal{N}$ and (ii) the pairwise geodesic distances between x_i and x_j for all i, j in $\mathcal{M}$.

Specifically, we can estimate $J_f G^{-1} J_f^\top \in \mathbb{R}^{n\times n}$ directly in local coordinates of $\mathcal{N}$. This estimation is derived from the following relationship between the Laplace-Beltrami operator $\Delta_{\mathcal{M}}$ and $J_f G^{-1} J_f^\top$ (Perraul-Joncas & Meila, 2013):

Proposition 2.1. *Let $\mathrm{f} : \mathcal{M} \to \mathbb{R}^n$ be a smooth mapping – where $\mathbb{R}^n$ is a local coordinate space for $\mathcal{N}$ –, represented in local coordinates by f. Then, at each $\mathrm{x} \in \mathcal{M}$,*

$$(J_f G^{-1} J_f^\top)_{lk} = \frac{1}{2}\Delta_{\mathcal{M}}\left(\mathrm{f}^l - \mathrm{f}^l(\mathrm{x})\right)\left(\mathrm{f}^k - \mathrm{f}^k(\mathrm{x})\right)|_{\mathrm{x}}, \quad (5)$$

where A_{lk} denotes (l,k)-th element of a matrix A and $\mathrm{f} = (\mathrm{f}^1, \ldots, \mathrm{f}^n)$.

To estimate $J_f G^{-1} J_f^\top$ by (5) from a finite set of points on $\mathcal{M}$, a sample-based approximation of $\Delta_{\mathcal{M}}$ is needed. For this, one first constructs an $N \times N$ kernel matrix

$$K = (K_{ij}),\ K_{ij} = \exp\left(-\frac{\mathrm{dist}(\mathrm{x}_i, \mathrm{x}_j)^2}{h}\right), \quad (6)$$

where $\mathrm{dist}(\mathrm{x}_i, \mathrm{x}_j)$ is the geodesic distance between points x_i and x_j, and h is a bandwidth parameter.

After computing $d_i = \sum_j K_{ij}$, $D = \mathrm{diag}(d_i)$, $\tilde{K} = D^{-1}KD^{-1}$, $\tilde{d}_i = \sum_j \tilde{K}_{ij}$ and $\tilde{D} = \mathrm{diag}(\tilde{d}_i)$, finally the *graph Laplacian* matrix $L \in \mathbb{R}^{N\times N}$ is obtained by

$$L = \frac{\tilde{D}^{-1}\tilde{K} - I}{h/4}, \quad (7)$$

with which the Laplace-Beltrami operator applied to a smooth function $\mathrm{q} : \mathcal{M} \to \mathbb{R}$ at each x_i can be approximated as $\Delta_{\mathcal{M}}\mathrm{q}\,(\mathrm{x}_i) = \sum_j L_{ij}\mathrm{q}(\mathrm{x}_j)$ (Hein et al., 2007). Then, as in Perraul-Joncas & Meila (2013) and Jang et al. (2021), one can discretize (5) to estimate $J_f G^{-1} J_f^\top$ at each x_i as

$$J_f G^{-1} J_f^\top = \frac{1}{2}\mathrm{f}(X)(\mathrm{diag}(L_i) - e_i e_i^\top L - L^\top e_i e_i^\top)\mathrm{f}(X)^\top, \quad (8)$$

where $\mathrm{f}(X) = [\mathrm{f}(\mathrm{x}_1), \cdots, \mathrm{f}(\mathrm{x}_N)] \in \mathbb{R}^{n\times N}$, $L_i \in \mathbb{R}^N$ is the i-th row of L, and $e_i \in \mathbb{R}^N$ is the i-th standard basis vector.

We note that $J_f G^{-1} J_f^\top$ estimated in this way depends only on the pairwise distances $\mathrm{dist}(\mathrm{x}_i, \mathrm{x}_j)$ and $z_i = \mathrm{f}(\mathrm{x}_i) \in \mathbb{R}^n$. To emphasize these dependencies, we denote $J_f G^{-1} J_f^\top$ estimated at $\mathrm{x_i}$ in (8) as follows:

$$\tilde{H}_i(L, \mathrm{f}(X)) := \tilde{H}(e_i, L, \mathrm{f}(X)) \in \mathbb{R}^{n\times n}, \quad (9)$$

where the graph Laplacian L, constructed by (6) and (7), only depends on the pairwise distances $\mathrm{dist}(\mathrm{x}_i, \mathrm{x}_j)$.

3. Graph Geometry-Preserving Autoencoder

We develop a novel autoencoder regularization framework that preserves the graph geometry. We begin this section by introducing notations and assumptions used throughout. Given a set of data points $X = \{\mathrm{x}_i\}_{i=1}^N$ where $\mathrm{x}_i \in \mathbb{R}^D$, we assume that (i) these data points approximately lie on an m-dimensional manifold $\mathcal{M} \subset \mathbb{R}^D$ ($m < D$) and (ii) we are given a graph $\mathcal{G} = (X, E)$ with N vertices X and edges $E = \{e_{ij}\}$ where e_{ij} represents the semantic distance between x_i and x_j (not all vertices need to be connected). We assume that there is an underlying Riemannian metric for $\mathcal{M}$ so that e_{ij} is approximately a geodesic distance between x_i and x_j.

As in the standard autoencoder framework, we will consider an encoder $\mathrm{f}_\theta : \mathbb{R}^D \to \mathcal{Z}$ and a decoder $\mathrm{g}_\phi : \mathcal{Z} \to \mathbb{R}^D$, each of which is approximated with a deep neural network. $\mathcal{Z}$ is the latent space, which we assume as $\mathcal{Z} = \mathbb{R}^m$ assigned with the identity metric $H(z) = I$. We are particularly interested in the encoder mapping restricted to the manifold $\mathcal{M}$, denoted by $\tilde{\mathrm{f}}_\theta := \mathrm{f}_\theta|_{\mathcal{M}} : \mathcal{M} \to \mathcal{Z}$. Our objective is to minimize the distortion of $\tilde{\mathrm{f}}_\theta$ between the Riemannian manifold $\mathcal{M}$ – whose Riemannian geometry is approximated with a graph $\mathcal{G}$ – and the Euclidean latent space $\mathcal{Z}$.

3.1. Coordinate-Free Approximation of Distortion of $\tilde{\mathrm{f}}_\theta$ using Graph $\mathcal{G}$

In this section, we present a method to approximately compute the distortion of $\tilde{\mathrm{f}}_\theta : \mathcal{M} \to \mathcal{Z}$, given a graph $\mathcal{G}$ whose edge values represent geodesic distances in the Riemannian manifold $\mathcal{M}$. A naive approach would be to compute the distortion measure of $\tilde{\mathrm{f}}_\theta : \mathcal{M} \to \mathcal{Z}$ by approximating (3) or (4) on the finite dataset X. However, since the underlying data manifold $\mathcal{M}$ is unknown during training time, we neither have local coordinates for $\mathcal{M}$ nor a Riemannian metric G expressed in these coordinates. Therefore, it is challenging to compute the distortion of $\tilde{\mathrm{f}}_\theta$ directly by (3) or (4).

Inspired by $J_f G^{-1} J_f^\top$ estimation introduced in Section 2.3, we introduce a coordinate-free approximation method to compute the distortion of $\tilde{\mathrm{f}}_\theta$, without the need for defining local coordinates and metric for $\mathcal{M}$, directly leveraging information in the given graph. First, we need pairwise geodesic distances between data points x_i and x_j. When only a sparse graph $\mathcal{G}$ of semantic distance is given, we use the *shortest-path distance* $\mathrm{dist}_{\mathcal{G}}(\mathrm{x}_i, \mathrm{x}_j)$ from x_i to x_j along $\mathcal{G}$ as an approximation of the underlying (but unknown) geodesic distance $\mathrm{dist}(\mathrm{x}_i, \mathrm{x}_j)$ in constructing the kernel matrix $K \in \mathbb{R}^{N\times N}$ (6).

Subsequently, we construct the graph Laplacian $L \in \mathbb{R}^{N\times N}$ as in (7). Then we can approximate $J_f G^{-1} J_f^\top$ with (9) and

rewrite (4) as a function of L and $\tilde{\mathrm{f}}_\theta$ as follows:

$$\mathcal{F}(\tilde{\mathrm{f}}_\theta, L) := \frac{1}{N}\sum_{i=1}^{N} \mathrm{Tr}\left[\tilde{H}_i(L, \tilde{\mathrm{f}}_\theta(X))^2 - 2\tilde{H}_i(L, \tilde{\mathrm{f}}_\theta(X))\right], \tag{10}$$

where μ in (4) is a probability measure for the empirical data distribution and $\tilde{\mathrm{f}}_\theta(X) := [\tilde{\mathrm{f}}_\theta(\mathrm{x}_1), \cdots, \tilde{\mathrm{f}}_\theta(\mathrm{x}_N)] \in \mathbb{R}^{n\times N}$. Notice that $\tilde{H}_i(L, \tilde{\mathrm{f}}_\theta(X))$ and hence the distortion measure $\mathcal{F}(\tilde{\mathrm{f}}_\theta, L)$ only depend on L and $\tilde{\mathrm{f}}_\theta(X)$. As a result, the distortion measure (10) does not require local coordinates x nor metric $G(x)$ of the data manifold $\mathcal{M}$. Additionally, we note that $\mathcal{F}(\tilde{\mathrm{f}}_\theta, L) = \mathcal{F}(\mathrm{f}_\theta, L)$ since $\tilde{\mathrm{f}}_\theta(X) = \mathrm{f}_\theta(X)$.

An autoencoder with a loss function augmented with the coordinate-free distortion measure (10) is referred to as **Graph Geometry-Preserving Autoencoder (GGAE)**:

$$\mathcal{L}_{GGAE} = \mathcal{L}_{recon}(\theta, \phi) + \alpha\mathcal{F}(\mathrm{f}_\theta, L), \tag{11}$$

where we use the standard mean squared reconstruction error for $\mathcal{L}_{recon}(\theta, \phi) := \frac{1}{N}\sum_{i=1}^{N} \|\mathrm{x}_i - (\mathrm{g}_\phi \circ \mathrm{f}_\theta)(\mathrm{x}_i)\|^2$ and $\alpha > 0$ is a weight parameter.

3.2. Mini-Batch Approximation of $JG^{-1}J^\top$

When training GGAE on a large dataset, computing the distortion measure (10) and its gradient at every iteration of gradient descent is computation- and memory-intensive because the entire dataset X must be encoded into $\mathrm{f}_\theta(X)$ to compute $\tilde{H}_i(L, \mathrm{f}_\theta(X))$. In practice, we need some means to approximate $\tilde{H}_i$ on a mini-batch $B \subset X$ of $b := |B| \ll N$ data points by encoding only the data points in B into $\mathrm{f}_\theta(B) \in \mathbb{R}^{n\times b}$. Our core idea is to construct – instead of the entire Laplacian matrix $L \in \mathbb{R}^{N\times N}$ – a mini-batch Laplacian matrix $L_B \in \mathbb{R}^{b\times b}$ to approximate $\tilde{H}_i = \tilde{H}_i(L_B, \mathrm{f}_\theta(B))$ on the mini-batch via (9). Then, we can finally compute a mini-batch distortion measure $\frac{1}{b}\sum_{\mathrm{x}_i\in B} \mathrm{Tr}\left[\tilde{H}_i^2 - 2\tilde{H}_i\right]$ as in (10). Hence in the following, we propose two methods to construct L_B, which we refer to as *Laplacian-Slicing* and *Batch-Kernel*, where L_B is denoted by L_B^s and L_B^k for distinction, respectively.

Laplacian-Slicing directly constructs $L_B^s \in \mathbb{R}^{b\times b}$ by choosing the $b \times b$ submatrix of L corresponding to the b data points in B. Afterwards, a mini-batch approximation of $\tilde{H}_i$ is obtained by (9) using inputs L_B^s and $\mathrm{f}_\theta(B)$, but with some correction terms (see Appendix A.1) to ensure each element of $\tilde{H}_i(L_B^s, \mathrm{f}_\theta(B))$ is an unbiased estimator of the corresponding element of $\tilde{H}_i(L, \mathrm{f}_\theta(X))$. It is noteworthy that L needs to be pre-computed only once before training as it is independent of the network parameters.

Batch-Kernel first constructs a mini-batch kernel matrix $K_B \in \mathbb{R}^{b\times b}$ by choosing the $b \times b$ submatrix of K corresponding to B. $L_B^k \in \mathbb{R}^{b\times b}$ is then obtained by (7), with $\tilde{K}, \tilde{D} \in \mathbb{R}^{b\times b}$ computed from K_B instead of the entire kernel matrix $K \in \mathbb{R}^{N\times N}$. A mini-batch approximation of $\tilde{H}_i$ is obtained by (9) using inputs L_B^k and $\mathrm{f}_\theta(B)$. We note that although only the points in the mini-batch are considered in the *Batch-Kernel* method, it effectively uses the shortest-path distances pre-computed on the entire graph by selecting the submatrix K_B from K. Like L in the *Laplacian-Slicing* method, K needs to be pre-computed only once before training as it is independent of the network parameters.

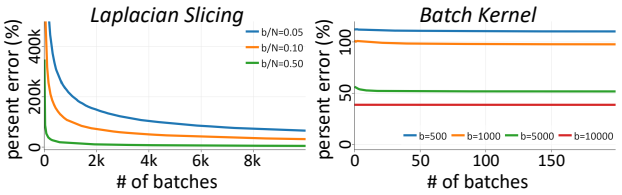

Figure 1. The plot displays the relationship between the number of sampled batches and the expectation error of two methods: *Laplacian-Slicing* (left) and *Batch-Kernel* (right).

Yet, $\tilde{H}_i(L_B^k, \mathrm{f}_\theta(B))$ is not necessarily an unbiased estimator of $\tilde{H}_i(L, \mathrm{f}_\theta(X))$. A theoretical support of *Batch-Kernel* lies in that a mini-batch of b data points can also be considered a set of b samples on the data manifold. Thus $\tilde{H}_i(L_B^k, \mathrm{f}_\theta(B))$ can be directly interpreted as a discrete approximation of $J_{f_\theta}G^{-1}J_{f_\theta}^\top$ at x_i. According to Hein et al. (2007), as $h \to 0$ and $bh^{\frac{m}{2}+1}/\log b \to \infty$, the discrete operator represented by the graph Laplacian matrix L_B^k converges (almost surely) to the Laplace-Beltrami operator $\Delta_\mathcal{M}$ in the sense that $\lim_{b\to\infty}\sum_j (L_B^k)_{ij}\mathrm{q}(\mathrm{x}_j) = \Delta_\mathcal{M}\mathrm{q}(\mathrm{x}_i)$ for a smooth function $\mathrm{q} : \mathcal{M} \to \mathbb{R}$, and the sample-based approximation error for each element of $J_{f_\theta}G^{-1}J_{f_\theta}^\top$ is given by

$$|(J_{f_\theta}G^{-1}J_{f_\theta}^\top)_{jk} - (\tilde{H})_{jk}| = O\left(\sqrt{h}\right) + O\left(\sqrt{\frac{\log b}{bh^{\frac{m}{2}+1}}}\right), \tag{12}$$

provided that $\tilde{\mathrm{f}}_\theta : \mathcal{M} \to \mathbb{R}^m$ is smooth.

Despite the unbiased estimation of $\tilde{H}_i(L, \mathrm{f}_\theta(X))$, we empirically observe that GGAE trained with *Laplacian-Slicing* fails to preserve geometry due to the high variance of the estimator $\tilde{H}_i(L_B^s, \mathrm{f}_\theta(B))$. We conduct a simple numerical experiment to investigate this further (see Appendix A.2 for details). We average the estimates for N_B randomly sampled mini-batches from the *Laplacian-Slicing* method as $\frac{1}{N_B}\sum_{l=1}^{N_B}\tilde{H}_i(L_{B_l}^s, \mathrm{f}_\theta(B_l))$ and test how quickly the average converges to $\tilde{H}_i(L, \mathrm{f}_\theta(X))$ as N_B increases. Figure 1 (*Left*) shows the percent error between $\tilde{H}_i(L, \mathrm{f}_\theta(X))$ and the averaged estimation as a function of N_B. The error appears to converge to zero as $N_B \to \infty$, however, its convergence rate is notably slow, maintaining a significantly high percent error even after sampling 10k batches. This indicates the unsuitability of the *Laplacian-Slicing* method for training GGAE.

In contrast, GGAE with *Batch-Kernel* successfully learns geometry-preserving representations despite the estimation bias, owing to low variance and reasonable scale of bias

from the ground truth $J_{f_\theta} G^{-1} J_{f_\theta}^\top$. Figure 1 (*Right*) shows the percent error measured between $J_{f_\theta} G^{-1} J_{f_\theta}^\top$ and its averaged estimate $\frac{1}{N_B} \sum_{l=1}^{N_B} \tilde{H}_i(L_{B_l}^k, \mathrm{f}_\theta(B_l))$ of *Batch-Kernel*. For a fixed mini-batch size $b = |B_l|$, the error quickly converges to a certain non-zero lower bound as N_B increases, indicating low variance. Moreover, this lower bound gradually decreases and converges to zero as the batch size b increases, which is consistent with the theoretical support discussed previously (12).

Overall, *Batch-Kernel* shows better estimates for $J_{f_\theta} G^{-1} J_{f_\theta}^\top$ and superior performance in graph geometry preservation than *Laplacian-Slicing*. Thus, we always use *Batch-Kernel* in the subsequent experiments.

4. Related Work

4.1. Geometry-Preserving Autoencoders

While vanilla autoencoders often produce geometrically distorted latent spaces, recent regularization approaches aim to learn geometry-preserving representations (Chen et al., 2020; Lee et al., 2022b; Nazari et al., 2023). Chen et al. (2020); Lee et al. (2022b) attempt to preserve scaled distances and angles, while Nazari et al. (2023) aim to learn volume-preserving representations. While these methods assume the identity metrics for the ambient data spaces, other works exploit non-Euclidean metrics (Lee et al., 2022a; Lee, 2024). We note that these works focus on minimizing the *distortions of decoders*.

One may question whether we can extend the existing geometry-preserving autoencoders to the setting where we are given a graph $\mathcal{G}$. To achieve this, we need a method for constructing a Riemannian metric for the high-dimensional ambient space from the graph. Some studies have attempted to estimate the ambient space metric (Alipanahi et al., 2008; Arvanitidis et al., 2020); however, estimating a metric for high-dimensional space is computation- and memory-intensive. Therefore, many studies resort to simplifying assumptions about the metric, such as assuming a diagonal matrix (Arvanitidis et al., 2020). Our method, in contrast, does not require ambient space metrics, minimizing the *distortions of encoders* directly approximated from the graphs.

4.2. Graph-based Autoencoders

There are regularization methods that leverage information from a given graph to enhance certain aspects of autoencoders (Singh & Nag, 2021; Lee et al., 2021; Moor et al., 2020). While NRAE (Lee et al., 2021) and TopoAE (Moor et al., 2020) focus on learning accurate manifolds and preserving topological structures, respectively, SPAE (Singh & Nag, 2021) attempts to find a graph geometry-preserving representation aligned with our objective.

The SPAE first computes the geodesic distances between all pairs of nodes in the graph and attempts to preserve these *global distances* in the latent space. Therefore, if these global distances are inaccurate, the performance degrades significantly. On the other hand, our method relies on the *local metrics*, thereby being robust to the global distance errors in the graphs.

5. Experimental Results

In Section 5.1, we introduce baselines and evaluation metrics. Section 5.2 compares GGAE with the baselines, using diverse datasets. Section 5.3 shows the advantageous use of GGAE's graph geometry-preserving latent representation in learning latent dynamics models. Lastly, in Section 5.4, we show the robustness of GGAE to missing edges in the graphs.

5.1. Experimental Settings

Baseline Models: The baseline models selected for comparison include AE, VAE (Kingma & Welling, 2013), geometric regularization methods such as IRAE (Lee et al., 2022b), GeomAE (Nazari et al., 2023), and graph-based methods such as SPAE (Singh & Nag, 2021), TopoAE (Moor et al., 2020), NRAE (Lee et al., 2021), and GRAE (Duque et al., 2022). The SPAE, TopoAE, and NRAE can be straightforwardly modified to use geometric information from a (non-Euclidean) graph, which we refer to as TopoAE-graph, SPAE-graph, and NRAE-graph, respectively.

Evaluation Metrics: We require evaluation metrics to measure (i) the accuracy of the learned manifold and (ii) the preservation of graph geometry in the learned latent space. The accuracy of the learned manifold is measured by the mean squared reconstruction error. To evaluate the preservation of graph geometry, we use *kNN*, *Spear*, $KL_{0.01}$, $KL_{0.1}$, and KL_1, adopted from Moor et al. (2020); Nazari et al. (2023). Originally, these metrics measure how well *Euclidean* data distances are preserved in the latent space. We adapt them by replacing the Euclidean data distances with the ground-truth *geodesic* distances in a given graph. *kNN* and $KL_{0.01}$ evaluate geometry preservation on a local scale, while *Spear* and KL_1 assess it on a global scale. $KL_{0.1}$ serves as an intermediary metric, balancing local and global geometry preservation. To evaluate latent dynamics learning performance, we use mean squared error between predicted images and actual future images, denoted as *dyn.*. For more details about the metrics, we refer the readers to Appendix B.2.

5.2. Graph Geometry-Preserving Representation

Table 1 presents a summary of our quantitative evaluation results. We rank the evaluation metrics for our methods and

Table 1. Average ranks of evaluation metrics measured using all datasets; the lower, the better. Exact values of metrics can be found in Appendix B.4.

	recon.	kNN	Spear	$KL_{0.01}$	$KL_{0.1}$	KL_1	dyn.
GGAE	7.2 ± 3.5	$\mathbf{1.5 \pm 0.6}$	$\mathbf{1.2 \pm 0.5}$	$\mathbf{1.2 \pm 0.5}$	$\mathbf{1.2 \pm 0.5}$	$\mathbf{1.2 \pm 0.5}$	**1**
SPAE-graph	5.2 ± 3.2	1.8 ± 1.0	1.8 ± 1.0	3.2 ± 3.2	3.2 ± 2.6	2.8 ± 2.2	3
TopoAE-graph	7.0 ± 2.6	4.0 ± 2.4	4.8 ± 1.9	5.5 ± 2.6	4.0 ± 3.4	5.0 ± 3.4	5
NRAE-graph	6.0 ± 4.2	8.0 ± 2.4	8.0 ± 3.4	9.2 ± 3.5	9.0 ± 2.4	8.4 ± 3.0	2
AE	4.8 ± 4.5	7.8 ± 2.4	7.8 ± 2.2	7.8 ± 2.2	8.2 ± 2.2	9.0 ± 1.8	10
VAE	4.7 ± 3.8	10.0 ± 1.0	9.7 ± 1.5	9.0 ± 1.7	10.0 ± 0.0	8.3 ± 2.1	-
IRAE	5.0 ± 2.4	6.0 ± 2.6	6.2 ± 2.6	5.2 ± 1.3	7.0 ± 1.8	6.8 ± 2.1	4
GeomAE	6.2 ± 2.6	7.0 ± 2.2	6.5 ± 2.9	6.0 ± 3.3	5.8 ± 2.8	5.8 ± 3.2	9
SPAE	$\mathbf{4.2 \pm 4.6}$	8.0 ± 2.8	8.8 ± 1.0	7.4 ± 1.8	7.8 ± 1.7	8.2 ± 1.3	7
TopoAE	8.2 ± 1.9	4.8 ± 1.7	4.5 ± 1.3	4.6 ± 2.6	4.0 ± 0.8	4.5 ± 0.6	8
GRAE	5.8 ± 2.1	6.8 ± 2.9	6.0 ± 2.6	6.2 ± 2.4	5.5 ± 1.7	5.4 ± 2.9	6

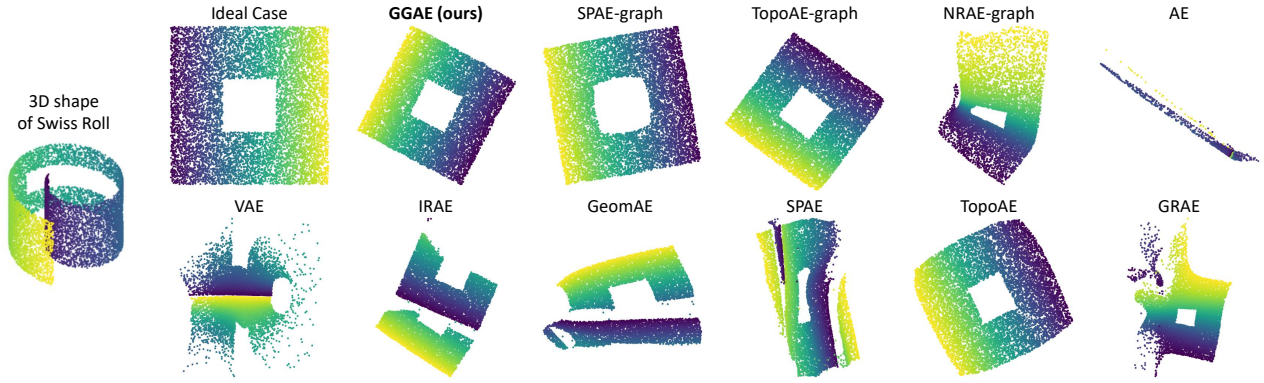

Figure 2. Top-left: An ideal case where Swiss Roll data points are encoded equidistantly. *Others:* The reconstruction results and a two-dimensional latent representation of our method, GGAE, and other baselines.

baselines, calculating the average across all datasets (details are given in the subsequent sections). The best results are indicated in bold. We fine-tune all models to achieve similarly low reconstruction errors; therefore, the high-rank result of GGAE in *recon.* does not carry significant meaning. GGAE achieves the best average ranks in key metrics related to the preservation of graph geometry.

5.2.1. Swiss Roll

Dataset and Graph: The Swiss Roll dataset consists of randomly sampled points on a two-dimensional, spiral-shaped manifold in $\mathbb{R}^3$, as shown in Figure 2. The geodesic distance between two data points is defined as the shortest path along the roll surface. We assume the Swiss Roll manifold inherits the Euclidean geometry of the ambient space $\mathbb{R}^3$. The inherited geometry can be discretely approximated by a k-nearest neighbor graph connecting the neighboring data points by their Euclidean distances. A notable feature of this dataset is a hole in the middle, which affects the structure of the neighboring graph; the shortest path on the graph is forced to detour around this hole. This results in a discrepancy between the shortest path distance on the graph and the global geodesic distance.

Results: Figure 2 shows two-dimensional latent representations; evaluation metrics are reported in Appendix B.4.1. Most methods fail to learn the correct connectivity of the manifold, while GGAE, SPAE-graph, TopoAE, and TopoAE-graph provide good results. Note that SPAE-graph generates a distorted hole caused by errors in global distances, specifically exaggerated shortest-path distances around the hole.

5.2.2. dSprites

Dataset and Graph: The dSprites dataset (Matthey et al., 2017) consists of 2D synthetic images generated from six ground truth factors of variation; *color*, *shape*, *rotation*, *scale*, x and y positions of a 2D shape. We fix the first three factors to white, square, and zero, so that the dataset consists of squares with varying *(scale, x, y)*. We refer to this three-dimensional variable as *latent vector*. This dataset inherently lies on a three-dimensional manifold. A meaningful distance along the manifold can be defined by the distance between latent vectors, which differs from the image space Euclidean distance. We construct a data graph $\mathcal{G}$ by connecting k-nearest neighbors in *(scale, x, y)* space.

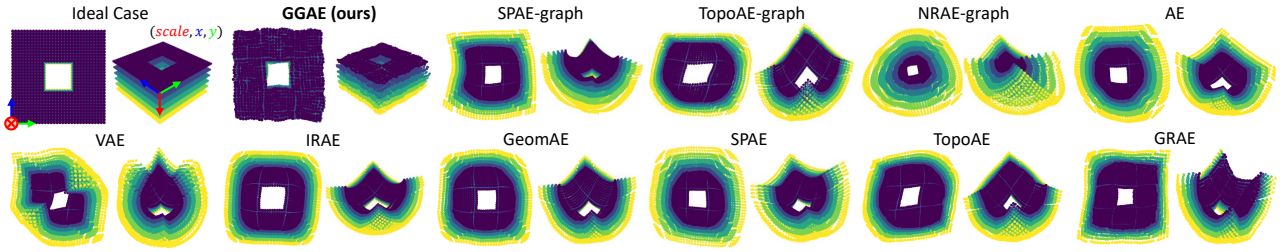

Figure 3. Top-left: An ideal representation that perfectly preserves the ground truth geometry of the dSprites manifold. *Others:* Three-dimensional latent representations learned by GGAE and other baselines, where brighter color indicates a larger value of the *scale* factor. Only GGAE learns a graph-preserving representation similar to the ideal case.

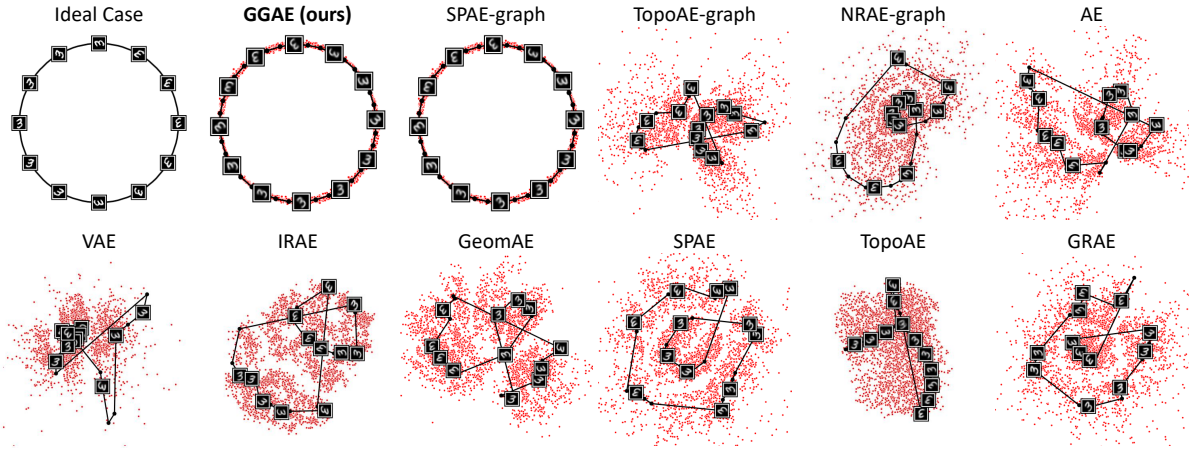

Figure 4. Top-left: An ideal case where a rotating sequence is encoded equidistantly. *Others:* Two-dimensional latent representation with frames from the same sequence annotated. Only the GGAE and SPAE-graph preserve the dynamical distance given by the graph.

Results Figure 3 shows three-dimensional latent representations; evaluation metrics are reported in Appendix B.4.2. We note that only GGAE learns a graph geometry-preserving representation similar to the ideal case (top-left in Figure 3). When training SPAE-graph, we encountered challenges in balancing the trade-off between reconstruction and regularization. The case with a low reconstruction error is presented in Figure 3; the opposite case can be found in Appendix B.4.2.

5.2.3. Rotating MNIST

Dataset and Graph: Our Rotating MNIST dataset consists of 36-frame videos of handwritten digit '3', with each frame created by rotating an image of '3' by 10° per step. The graph is constructed by connecting all pairs of consecutive frames, including the last and first frames.

Results: Figure 4 illustrates two-dimensional latent representations; evaluation metrics are reported in Appendix B.4.3. The correct encoding of the dataset's intrinsic rotational dynamics would be reflected by a loop-shaped structure, as shown in the *Top-left*. GGAE and SPAE-graph successfully preserve the rotational dynamics in the form of a loop. On the contrary, other baseline models produce entangled or disconnected representations, failing to adequately encode the sequential relationships between frames. In GGAE and SPAE-graph, to capture a loop-shaped structure, reconstruction is sacrificed to some extent, resulting in various shapes of 3's collapsing into a similar shape. To prevent this while enabling loop-shaped encoding, a higher-dimensional latent dimension may be needed.

5.3. Learning Latent Dynamics from Image Sequence

In this section, we demonstrate how graph geometry-preserving representations can benefit the downstream task of latent dynamics learning. Once autoencoders are trained, we fix them and train dynamics models in the latent spaces – that are trained, using time series data, to predict a sequence of future data given past observation data. We adopt the architecture from Hafner et al. (2019; 2020a;b); details can be found in Appendix B.3.

Dataset and Graph: We consider images containing a robot holding a block as shown in Figure 5 (*Left*). We construct a similarity graph of the images, by using the block's position, connecting images with similar block positions by edges in the graph, based on the idea that preserving this graph geometry would lead to good representations for learning the latent dynamics. We note that using the image Euclidean distances does not provide this graph.

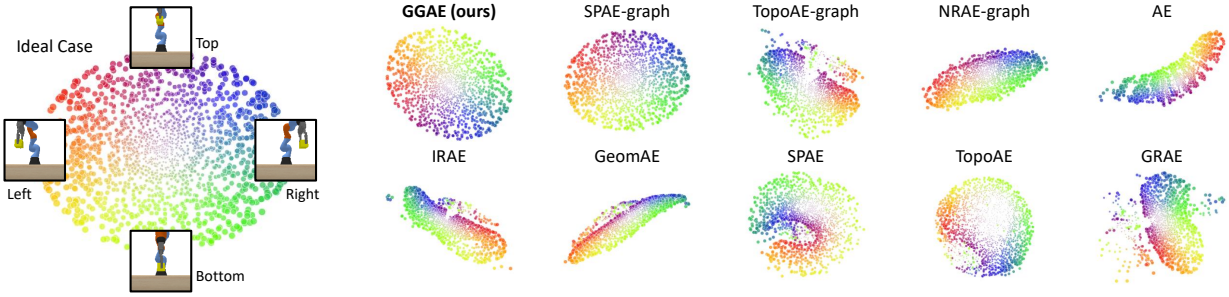

Figure 5. Left: An ideal case where images are encoded while preserving the similarity from the block position. *Others*: Two-dimensional latent representations of GGAE and other baselines. Only GGAE and SPAE-graph preserve the geometry of the similarity graph.

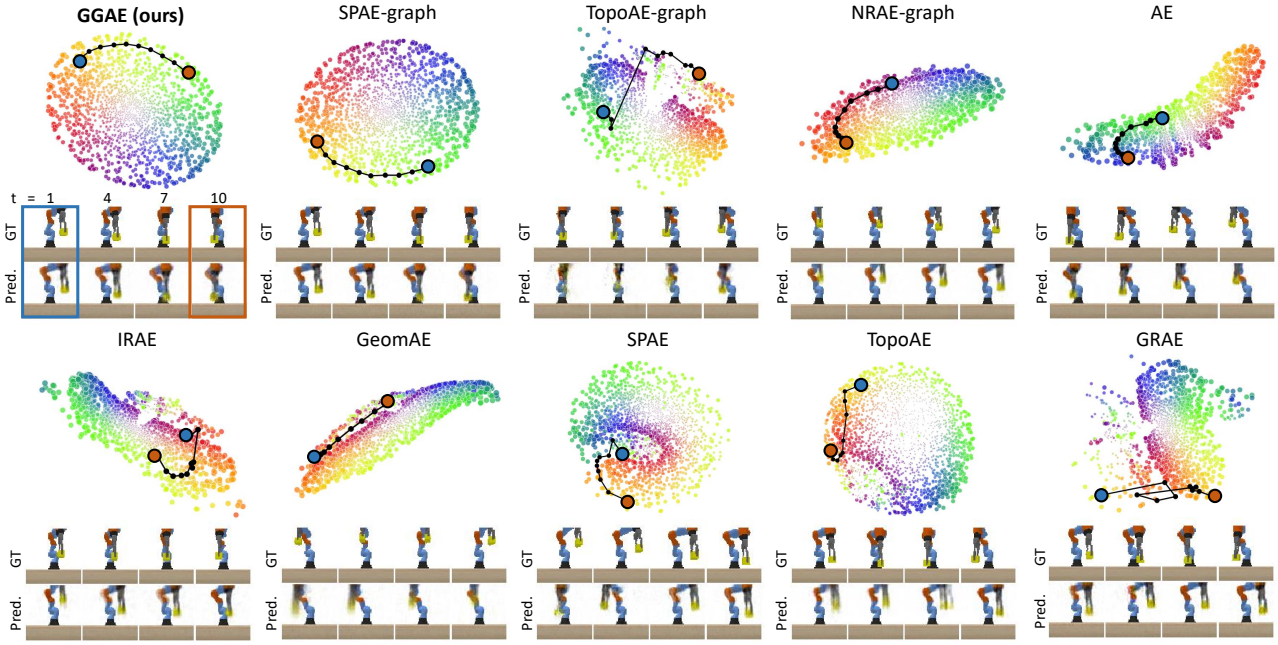

Figure 6. Ten future latent states (black dots) predicted by the trained dynamics model and corresponding reconstructed images at $t = 1, 4, 7, 10$. The dynamics model, when trained in a distorted latent space, exhibits irregular step sizes or jumps in the latent space, leading to its failure to accurately predict future images.

Results (graph geometry-preserving): The trained latent representations are shown in Figure 5. IRAE, GeomAE, SPAE, TopoAE, and GRAE show distorted or disconnected latent representations, indicating that the image space Euclidean metric is inappropriate. While TopoAE-graph and NRAE-graph fail, GGAE and SPAE-graph successfully preserve the graph geometry, showing latent representations nearly identical to the ideal case (*Left*).

Results (latent dynamics learning): To evaluate the performance of the latent dynamics models, we predict the next ten images. This process involves the latent dynamics model first predicting ten future states in the latent space, followed by the decoder reconstructing these states into images. The results are illustrated in Figure 6. Models trained with distorted or disconnected latent spaces predict sequences of latent states with non-uniform distances between consecutive states and abnormal jumps. In contrast, GGAE and SPAE-graph predict sequences of states that are smooth and evenly spaced, leading to the successful prediction of future images.

5.4. Robustness to Missing Edges: GGAE vs SPAE-graph

In reality, acquiring all pairwise similarities of vertices can be cost-intensive or infeasible. In this section, we compare the robustness of GGAE and SPAE-graph, which perform the best among the baselines, given a graph with missing edges. We disconnect some of the edges with a probability of $p > 0$; as the disconnecting probability p increases, the graph becomes sparser, resulting in a higher error between the shortest path distances on the graph and the actual geodesic distances. We focus on two examples from previous sections: (i) the Swiss Roll manifold and (ii) the images of a robot arm.

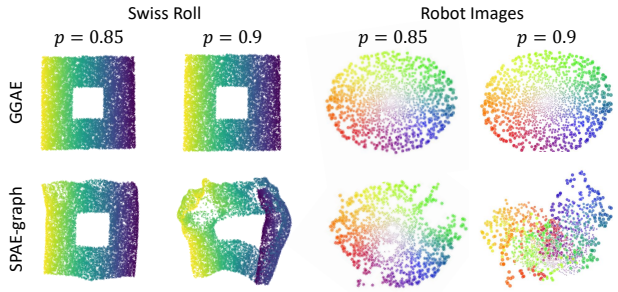

Figure 7. The trained latent representations of GGAE and SPAE-graph on the Swiss Roll and Robot Image dataset with sparse graphs. The graph geometry-preserving performance of SPAE-graph decreases significantly as the graph has fewer edges.

Figure 7 illustrates two-dimensional latent representations of GGAE and SPAE-graph with $p = 0.85, 0.9$. In both datasets, GGAE retains its graph geometry preservation with sparse data graphs, while the performance of SPAE-graph degrades drastically as the graph gets sparser. This demonstrates that GGAE's local metric-based regularization is more robust to missing edges compared to SPAE's global distance-based regularization, as missing edges can introduce errors in global distance computation. The quantitative evaluation results are in Appendix B.4.5, with latent space visualization in Figure 11.

6. Conclusion

In this study, we have proposed the Graph Geometry-Preserving Autoencoder (GGAE), a novel framework for autoencoder regularization. GGAE effectively preserves the data manifold geometry based on a given similarity graph, utilizing a Riemannian geometric distortion measure computed with a graph Laplacian as a regularizer. For efficient implementations of large-scale autoencoder training, we have devised two batch-based graph Laplacian approximation methods: (i) *Laplacian-Slicing* and (ii) *Batch-Kernel*. Through empirical error analysis, we have found that the *Batch-Kernel* method significantly outperforms *Laplacian-Slicing*. Our extensive experiments have confirmed that GGAE outperforms other geometry-preserving autoencoders, particularly showcasing robustness to errors in global geodesic distances on graphs. Additionally, a case study with high-dimensional images of a robot manipulator illustrates the effectiveness of GGAE in providing a suitable low-dimensional latent representations for latent dynamic learning.

One challenge of training GGAE, common among kernel-based methods, is the lack of a canonical strategy for tuning the bandwidth parameter, despite its significant impact on performance. The appropriate bandwidth scale is often related to the statistical distribution of the edge lengths in the graph, requiring independent bandwidth tuning for each dataset used in our experiments. For datasets without a predefined graph, various options for similarity graph must be explored, each of which requires finding a suitable bandwidth. This can be burdensome when applying GGAE to new datasets.

Acknowledgements

This work was supported in part by IITP-MSIT grant RS-2021-II212068 (SNU AI Innovation Hub), IITP-MSIT grant 2022-0-00480 (Training and Inference Methods for Goal-Oriented AI Agents), KIAT grant P0020536 (HRD Program for Industrial Innovation), ATC+ MOTIE Technology Innovation Program grant 20008547, SRRC NRF grant RS-2023-00208052, SNU-AIIS, SNU-IPAI, SNU-IAMD, SNU BK21+ Program in Mechanical Engineering, and SNU Institute for Engineering Research. Yonghyeon Lee was the beneficiary of an individual grant from CAINS supported by a KIAS Individual Grant (AP092701) via the Center for AI and Natural Sciences at Korea Institute for Advanced Study. Cheongjae Jang was supported in part by IITP-MSIT grant RS-2020-II201373 (Artificial Intelligence Graduate School Program for Hanyang University) and NRF-Ministry of Education grant RS-2023-00249714 (Basic Science Research Program).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Alipanahi, B., Biggs, M., Ghodsi, A., et al. Distance metric learning vs. fisher discriminant analysis. In *Proceedings of the 23rd national conference on Artificial intelligence*, volume 2, pp. 598–603, 2008.

Arvanitidis, G., Hansen, L. K., and Hauberg, S. Latent space oddity: on the curvature of deep generative models. *arXiv preprint arXiv:1710.11379*, 2017.

Arvanitidis, G., Hauberg, S., and Schölkopf, B. Geometrically enriched latent spaces. *arXiv preprint arXiv:2008.00565*, 2020.

Belkin, M. and Niyogi, P. Semi-supervised learning on riemannian manifolds. *Machine learning*, 56:209–239, 2004.

Belkin, M., Sun, J., and Wang, Y. Constructing laplace operator from point clouds in $\mathbb{R}^d$. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1031–1040. SIAM, 2009.

Chazal, F., Cohen-Steiner, D., and Mérigot, Q. Geometric inference for probability measures. *Foundations of Computational Mathematics*, 11:733–751, 2011.

Chen, N., Klushyn, A., Ferroni, F., Bayer, J., and Van Der Smagt, P. Learning flat latent manifolds with vaes. *arXiv preprint arXiv:2002.04881*, 2020.

Chong, Y., Ding, Y., Yan, Q., and Pan, S. Graph-based semi-supervised learning: A review. *Neurocomputing*, 408:216–230, 2020.

Coifman, R. R. and Lafon, S. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

Costa, J. A. and Hero, A. O. Manifold learning using euclidean k-nearest neighbor graphs [image processing examples]. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pp. iii–988. IEEE, 2004.

Duque, A. F., Morin, S., Wolf, G., and Moon, K. R. Geometry regularized autoencoders. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Eells, J. and Sampson, J. H. Harmonic mappings of riemannian manifolds. *American journal of mathematics*, 86(1): 109–160, 1964.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020a.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020b.

Hauberg, S., Freifeld, O., and Black, M. A geometric take on metric learning. *Advances in Neural Information Processing Systems*, 25, 2012.

Hein, M., Audibert, J.-Y., and Luxburg, U. v. Graph laplacians and their convergence on random neighborhood graphs. *Journal of Machine Learning Research*, 8(6), 2007.

Jang, C., Noh, Y.-K., and Park, F. C. A riemannian geometric framework for manifold learning of non-euclidean data. *Advances in Data Analysis and Classification*, 15 (3):673–699, 2021.

Jang, C., Lee, Y., Noh, Y.-K., and Park, F. C. Geometrically regularized autoencoders for non-euclidean data. In *The Eleventh International Conference on Learning Representations*, 2022.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kobak, D. and Berens, P. The art of using t-sne for single-cell transcriptomics. *Nature communications*, 10(1):5416, 2019.

Lee, Y. A geometric perspective on autoencoders. *arXiv preprint arXiv:2309.08247*, 2023.

Lee, Y. Mmp++: Motion manifold primitives with parametric curve models. *arXiv preprint arXiv:2310.17072*, 2024.

Lee, Y. and Park, F. C. On explicit curvature regularization in deep generative models. In *Topological, Algebraic and Geometric Learning Workshops 2023*, pp. 505–518. PMLR, 2023.

Lee, Y., Kwon, H., and Park, F. Neighborhood reconstructing autoencoders. *Advances in Neural Information Processing Systems*, 34:536–546, 2021.

Lee, Y., Kim, S., Choi, J., and Park, F. A statistical manifold framework for point cloud data. In *International Conference on Machine Learning*, pp. 12378–12402. PMLR, 2022a.

Lee, Y., Yoon, S., Son, M., and Park, F. C. Regularized autoencoders for isometric representation learning. In *International Conference on Learning Representations*, 2022b.

Matthey, L., Higgins, I., Hassabis, D., and Lerchner, A. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

Moor, M., Horn, M., Rieck, B., and Borgwardt, K. Topological autoencoders. In *International conference on machine learning*, pp. 7045–7054. PMLR, 2020.

Nazari, P., Damrich, S., and Hamprecht, F. A. Geometric autoencoders–what you see is what you decode. *arXiv preprint arXiv:2306.17638*, 2023.

Perraul-Joncas, D. and Meila, M. Non-linear dimensionality reduction: Riemannian metric estimation and the problem of geometric discovery. *arXiv preprint arXiv:1305.7255*, 2013.

Rosenberg, S. *The Laplacian on a Riemannian manifold: an introduction to analysis on manifolds*. Number 31. Cambridge University Press, 1997.

Shao, H., Kumar, A., and Thomas Fletcher, P. The riemannian geometry of deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 315–323, 2018.

Singh, A. and Nag, K. Structure-preserving deep autoencoder-based dimensionality reduction for data visualization. In *2021 IEEE/ACIS 22nd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pp. 43–48. IEEE, 2021.

A. Mini-Batch Approximation: *Laplacian-Slicing* vs *Batch-Kernel*

In this section, we present the mathematical formulation of *Laplacian-Slicing* along with a comparative analysis of *Laplacian-Slicing* and *Batch-Kernel*. Additionally, we offer an empirical analysis explaining the failure of *Laplacian-Slicing*, focusing on its convergence rate.

A.1. Mathematical Formulation of *Laplacian-slicing*

Unbiased Estimation of $\tilde{H}_i(L, \mathrm{f}(X))$: For a fixed data point x_i, the graph Laplacian $L \in \mathbb{R}^{N\times N}$ and $\mathrm{f}(X) \in \mathbb{R}^{n\times N}$, we can rewrite (8) from the main text as

$$\tilde{H}_i(L, \mathrm{f}(X))_{kl} = \sum_{m=1}^{N} \frac{1}{2} L_{im}\big(\mathrm{f}^k(\mathrm{x}_m)\mathrm{f}^l(\mathrm{x}_m) - \mathrm{f}^k(\mathrm{x}_i)\mathrm{f}^l(\mathrm{x}_m) - \mathrm{f}^k(\mathrm{x}_m)\mathrm{f}^l(\mathrm{x}_i)\big) \tag{13}$$

where A_{kl} denotes (k,l)-th element of a matrix A and $\mathrm{f} = (\mathrm{f}^1, ..., \mathrm{f}^n)$.

Define an $n \times n$ matrix $\tilde{H}_i^m$ such that $(\tilde{H}_i^m)_{kl} := \frac{1}{2} L_{im}\big(\mathrm{f}^k(\mathrm{x}_m)\mathrm{f}^l(\mathrm{x}_m) - \mathrm{f}^k(\mathrm{x}_i)\mathrm{f}^l(\mathrm{x}_m) - \mathrm{f}^k(\mathrm{x}_m)\mathrm{f}^l(\mathrm{x}_i)\big)$ for $k, l = 1, ..., n$. Then, we can rewrite (13) as

$$\tilde{H}_i(L, \mathrm{f}(X)) = \sum_{m=1}^{N} \tilde{H}_i^m. \tag{14}$$

Now, consider the set of all possible mini-batches $B \subset X$ that contains x_i. We will denote this set as $\mathcal{B}_i$. For a batch B in $\mathcal{B}_i$, the *Laplacian-Slicing* method approximates $\tilde{H}_i(L, \mathrm{f}(X))$ on B as

$$\tilde{H}_i(L_B^s, \mathrm{f}(B)) = \sum_{\mathrm{x}_m \in B} \tilde{H}_i^m, \tag{15}$$

where $L_B^s \in \mathbb{R}^{b\times b}$ is the submatrix of $L \in \mathbb{R}^{N\times N}$ corresponding to the data points in B.

The probability mass $P(B) : \mathcal{B}_i \to \mathbb{R}$ is uniformly $\frac{1}{\binom{N-1}{b-1}}$ for all B in $\mathcal{B}_i$. We then derive the expectation of (k,l)-th element of $\tilde{H}_i(L_B^s, \mathrm{f}(B))$ as

$$\begin{aligned} \mathop{\mathbb{E}}_{B\sim P(B)}\big[\tilde{H}_i(L_B^s, \mathrm{f}(B))\big] &= \sum_{B\in\mathcal{B}_i} P(B)\big[\tilde{H}_i(L_B^s, \mathrm{f}(B))\big] \\ &= \frac{1}{\binom{N-1}{b-1}} \sum_{B\in\mathcal{B}_i} \sum_{\mathrm{x}_m\in B} (\tilde{H}_i^m). \end{aligned} \tag{16}$$

In all $\binom{N-1}{b-1}$ batches in $\mathcal{B}_i$, x_i is sampled $\binom{N-1}{b-1}$ times (i.e., always) and other data points are sampled $\binom{N-2}{b-2}$ times each. So we can rewrite (16) as

$$\begin{aligned} \mathop{\mathbb{E}}_{B\sim P(B)}\big[\tilde{H}_i(L_B^s, \mathrm{f}(B))\big] &= \frac{1}{\binom{N-1}{b-1}}\left[\binom{N-1}{b-1}\tilde{H}_i^i + \binom{N-2}{b-2}\left(\sum_{m=1}^{i-1}\tilde{H}_i^m + \sum_{m=i+1}^{N}\tilde{H}_i^m\right)\right] \\ &= \tilde{H}_i^i + \frac{b-1}{N-1}\left(\sum_{m=1}^{N}\tilde{H}_i^m - \tilde{H}_i^i\right) \\ &= \frac{N-b}{N-1}\tilde{H}_i^i + \frac{b-1}{N-1}\sum_{m=1}^{N}\tilde{H}_i^m. \end{aligned} \tag{17}$$

From (14) and (17), the $\tilde{H}_i(L, \mathrm{f}(X))$ can be approximated from the expectation of mini-batch estimation as

$$\tilde{H}_i(L, \mathrm{f}(X)) = \mathop{\mathbb{E}}_{B\sim P(B)}\big[\frac{N-1}{b-1}\tilde{H}_i(L_B^s, \mathrm{f}(B)) - \frac{N-b}{b-1}\tilde{H}_i^i\big] \tag{18}$$

which means that each element of sample-based approximation $\hat{H}_{i,B} := \frac{N-1}{b-1}\tilde{H}_i(L_B^s, \mathrm{f}(B)) - \frac{N-b}{b-1}\tilde{H}_i^i$ is an unbiased estimator of the corresponding element of $\tilde{H}_i(L, \mathrm{f}(X))$ computed from the global Laplacian $L \in \mathbb{R}^{N\times N}$ and the entire dataset X.

Variance of Estimator: We now compute the variance of each element of the unbiased estimator obtained in (18). For simplicity, we denote the unbiased estimator by $\hat{H}_{i,B} := \frac{N-1}{b-1}\tilde{H}_i(L_B^s, \mathrm{f}(B)) - \frac{N-b}{b-1}\tilde{H}_i^i$. Since the second term does not depend on batch sampling, we compute the variance of (k,l)-th element of $\tilde{H}_i(L_B^s, \mathrm{f}(B))$.

Define indicator variables C_m $(m = 1, \ldots, i-1, i+1 \ldots N)$ to be the random variables that each takes the value of 1 if x_m is included in a batch $B \in \mathcal{B}_i$ and 0 otherwise. Then, $\tilde{H}_i(L_B^s, \mathrm{f}(B))_{kl}$ can be expressed as

$$\tilde{H}_i(L_B^s, \mathrm{f}(B))_{kl} = \sum_{m=1}^{N} (\tilde{H}_i^m)_{kl} = (\tilde{H}_i^i)_{kl} + \sum_{m\neq i} (\tilde{H}_i^m)_{kl} C_m. \tag{19}$$

When uniformly sampling $(b-1)$ out of $(N-1)$ points without replacement, it is known that the corresponding indicator variables, which are exactly the C_m's we defined, satisfy the following:

$$\mathrm{Var}(C_m) = p(1-p), \quad \mathrm{Cov}(C_{m_1}, C_{m_2}) = -\frac{2p(1-p)}{N-2}\ (m_1 \neq m_2), \tag{20}$$

where $p := \frac{b-1}{N-1}$ is the uniform probability for each point to be sampled.

Thus, the variance of (19) can be computed as

$$\mathrm{Var}\big[\tilde{H}_i(L_B^s, \mathrm{f}(B))_{kl}\big] = p(1-p)\sum_{m\neq i}((\tilde{H}_i^m)_{kl})^2 - \frac{2p(1-p)}{N-2}\sum_{m_1,m_2\neq i}(\tilde{H}_i^{m_1})_{kl}(\tilde{H}_i^{m_2})_{kl}. \tag{21}$$

Finally, the variance of (k,l)-th element of the sample-based approximation $\hat{H}_{i,B}$ is computed as

$$\mathrm{Var}\big[(\hat{H}_{i,B})_{kl}\big] = \frac{N-1}{b-1}\left(\sum_{m\neq i}((\tilde{H}_i^m)_{kl})^2 - \frac{2}{(N-2)}\sum_{m_1,m_2\neq i}(\tilde{H}_i^{m_1})_{kl}(\tilde{H}_i^{m_2})_{kl}\right). \tag{22}$$

For a fixed dataset (i.e., N and $(\tilde{H}_i^m)_{kl}$ both fixed), the variance increases with decreasing batch size b. This is consistent with the empirical results presented in the left subplot of Figure 9.

A.2. Empirical Error Analysis

We conduct experiments on the Swiss Roll and dSprites datasets to evaluate and compare two methods for constructing a mini-batch Laplacian matrix: *Batch-Kernel* and *Laplacian-Slicing*. The trained latent spaces for each model are depicted in Figure 8, and the corresponding evaluation metrics are reported in Table 2. In this table, the GGAE model using *Batch-Kernel* is labeled as GGAE-*k*, and the one using *Laplacian-Slicing* as GGAE-*s*. As illustrated in Figure 8, GGAE-*s* does not successfully preserve the geometry of the given graph in either dataset. The quantitative comparison results align with these observations, showing that GGAE-*k* surpasses GGAE-*s* across all evaluation metrics.

Table 2. Quantitative comparison of *Laplacian-Slicing* and *Batch-Kernel*.

dataset	model	recon.	kNN	Spear	$KL_{0.01}$	$KL_{0.1}$	KL_1
Swiss Roll	GGAE-*k*	**4.874e-3**	**0.9911**	**1.0000**	**3.673e-5**	**4.730e-6**	**5.030e-7**
	GGAE-*s*	9.407e-2	0.8306	0.8820	5.748e-2	9.265e-3	6.450e-4
dSprites	GGAE-*k*	**2.830e-4**	**0.9683**	**0.9979**	**1.633e-3**	**3.310e-4**	**1.291e-5**
	GGAE-*s*	6.372e-4	0.5780	0.4252	3.416e-1	1.052e-1	4.691e-3

To understand why *Laplacian-Slicing* fails despite the validity of (18), we empirically examined the convergence and its rate as indicated in (18). Assuming that $\mathrm{f}: \mathbb{R}^3 \to \mathbb{R}^2$ is an ideal isometric mapping for the Swiss Roll, we analyzed whether the average of $\frac{N-1}{b-1}\tilde{H}_i(L_B^s, \mathrm{f}(B)) - \frac{N-b}{b-1}\tilde{H}_i^i$ converges to $\tilde{H}_i(L, \mathrm{f}(X))$ for randomly sampled batches, and if so, how quickly this convergence occurs.

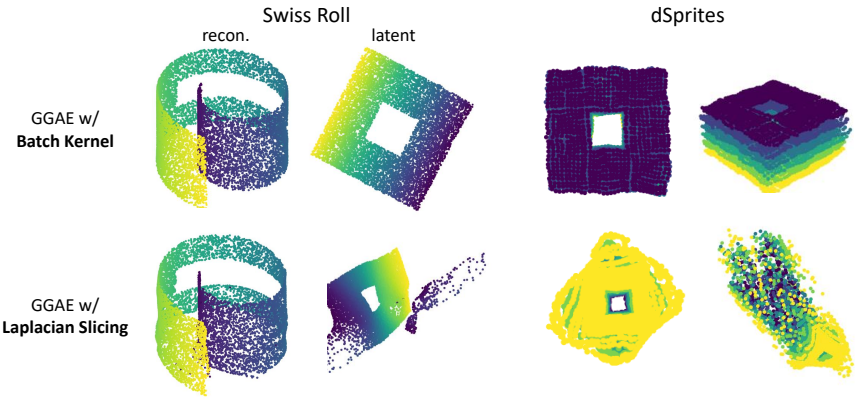

Figure 8. The experimental results of GGAEs using *Batch-Kernel* and *Laplacian-Slicing* on the Swiss Roll and dSprites datasets.

Using k randomly sampled batches, we calculate the error in the expectation from *Laplacian-Slicing* as

$$\text{percent error (\%)} = \frac{||\tilde{H}_i(L, \mathrm{f}(X)) - \frac{1}{k}\sum_{B=B_1}^{B_k}\left[\frac{N-1}{b-1}\tilde{H}_i(L_B^s, \mathrm{f}(B)) - \frac{N-b}{b-1}\tilde{H}_i^i\right]||_F}{||\tilde{H}_i(L, \mathrm{f}(X))||_F} \times 100. \tag{23}$$

where $||\cdot||_F$ denotes the Frobenius norm [2]. We also calculate the error of *Batch-Kernel* as

$$\text{percent error (\%)} = \frac{||(JG^{-1}J^\top)_{\mathrm{x_i}} - \frac{1}{k}\sum_{B=B_1}^{B_k}\tilde{H}_i(L_B^k, \mathrm{f}(B))||_F}{||(JG^{-1}J^\top)_{\mathrm{x_i}}||_F} \times 100. \tag{24}$$

Here, $(JG^{-1}J^\top)_{\mathrm{x_i}}$ denotes $JG^{-1}J^\top$ at $\mathrm{x_i}$.

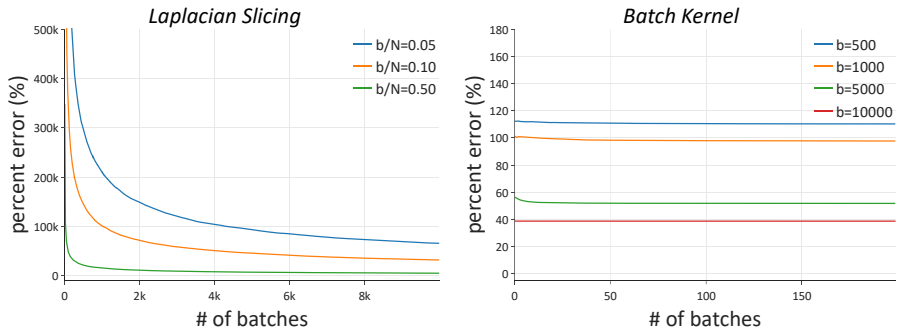

Figure 9. The plot displays the relationship between the number of sampled batches and the expectation error of two methods: *Laplacian-Slicing* (left) and *Batch-Kernel* (right).

Figure 9 (*Left*) illustrates the percent error between $\tilde{H}_i(L, \mathrm{f}(X))$ and $\frac{1}{k}\sum_{B=B_1}^{B_k}\left[\frac{N-1}{b-1}\tilde{H}_i(L_B^s, \mathrm{f}(B)) - \frac{N-b}{b-1}\tilde{H}_i^i\right]||_F$ as a function of the number of sampled batches k. Although the error converges toward zero as k approaches infinity, the rate of convergence is slow, with a notably high percent error persisting even after 10k batch samples. Figure 9 (*Right*) displays the percent error of the *Batch-Kernel* method measured between the actual $JG^{-1}J^\top$ and its averaged estimate $\frac{1}{k}\sum_{B=B}^{B_k}\tilde{H}_i(L_B^k, \mathrm{f}(B))$. At a constant batch size $b = |B|$, the error quickly reaches a certain non-zero minimum as k increases. As the batch size b grows, the error gradually diminishes and eventually zeroes out, consistent with the theoretical support previously discussed (12). Comparatively, the *Batch-Kernel*'s percent error is substantially lower than that of *Laplacian-Slicing*, demonstrating the superior performance of the *Batch-Kernel* method.

[2]One might wonder why the geodesic distances between two symmetric positive-definite matrices, derived as $d(P_1, P_2) = (\sum_{i=1}^n(\log\lambda_i(P_1^{-1}P_2))^2)^{1/2}$, are not used to measure the difference between $\tilde{H}_i(L, \mathrm{f}(X))$ and its expectation. The reason is that we observed, in the middle of convergence, the term $\frac{1}{k}\sum_{B=B_1}^{B_k}\left[\frac{N-1}{b-1}\tilde{H}_i(L_B^s, \mathrm{f}(B)) - \frac{N-b}{b-1}\tilde{H}_i^i\right]$ is not a symmetric positive-definite matrix. Therefore, we instead employ the Frobenius norm to calculate the error.

A.3. Pseudocode

In this section, we provide python-style pseudocode for *Batch-Kernel* and *Laplacian-Slicing*. *Batch-Kernel* chooses a submatrix of the kernel matrix to compute the batch Laplacian, whereas *Laplacian-Slicing* computes the Laplacian first and then chooses a submatrix of the Laplacian matrix. The functions `Equation7`, `Equation8`, and `Equation18` represent the corresponding equations from the main text.

```
def BatchKernel(K, fX, batch_indices, bandwidth):
    # ----------- INPUT -----------
    # K             : a kernel matrix, size of (N, N)
    # fX            : embedded points f(X), size of (N, n)
    # batch_indices : indices of data points in a batch, size of (B)
    # bandwidth     : a bandwidth parameter, size of (1)

    # ----------- OUTPUT ----------
    # H_tilde       : J_f G^{-1} J_f^T for points in a batch, size of (B, n, n)

    # Choose a submatrix of the kernel matrix K
    K_B = K[batch_indices, :][:, batch_indices]     # size of (B, B)

    # Batch the embedded points
    fB = fX[batch_indices, :]                       # size of (B, n)

    # Calculate Equation (7): calculate the batch Laplacian
    L_B = Equation7(K_B, bandwidth)                 # size of (B, B)

    # Calculate Equation (8): calculate J_f G^{-1} J_f^T
    H_tilde = Equation8(L_B, fB)                    # size of (B, n, n)

    return H_tilde
```

Listing 1. pseudocode for *Batch-Kernel*

```
def LaplacianSlicing(K, fX, batch_indices, bandwidth):
    # ----------- INPUT -----------
    # K             : a kernel matrix, size of (N, N)
    # fX            : embedded points f(X), size of (N, n)
    # batch_indices : indices of data points in a batch, size of (B)
    # bandwidth     : a bandwidth parameter, size of (1)

    # ----------- OUTPUT ----------
    # H_tilde       : J_f G^{-1} J_f^T for points in a batch, size of (B, n, n)

    # Calculate Equation (7): calculate the Laplacian
    L = Equation7(K, bandwidth)                     # size of (N, N)

    # Choose a submatrix of the Laplacian matrix
    L_B = L[batch_indices, :][:, batch_indices]     # size of (B, B)

    # Batch the embedded points
    fB = fX[batch_indices, :]                       # size of (B, n)

    # Calculate Equation (18) in Appendix A.1: calculate J_f G^{-1} J_f^T
    H_tilde = Equation18(L_B, fB)                   # size of (B, n, n)

    return H_tilde
```

Listing 2. pseudocode for *Laplacian-Slicing*

B. Further Experimental Details

B.1. Computational Complexity

GGAE initially computes the shortest path distances between every node on a given graph before training. Then, the distortion measure is approximated from the mini-batch kernel matrix by (6)∼(10) at every training iteration. This involves multiplications of $b \times b$ and $b \times n$ matrices, where b is the batch size and n the latent dimension. The computational complexity of our distortion measure is $O(b^3 + b^2 n)$. In our experiments, this complexity does not pose a significant computational burden during training. The training time measurements of GGAE and the baselines are presented in Table 3. The experiments utilized an i9-10900K CPU and a Geforce RTX 4090 GPU. For the dataset, we employed the dSprites dataset used in Section 5.2.2, with a batch size of 100. With this setup, we measured the time taken for 100 training steps. GGAE exhibits comparable computation time to the baselines utilizing simple regularization terms, such as AE, VAE, SPAE, and GRAE. Furthermore, GGAE demonstrates shorter computation time relative to models employing Jacobian-based regularization terms (IRAE, GeomAE) or other computationally intensive regularization terms (TopoAE, NRAE).

Table 3. The training time measurements of GGAE and baselines

	Elapsed time (s)
GGAE	0.487 ± 0.007
SPAE-graph	0.456 ± 0.014
TopoAE-graph	1.401 ± 0.067
NRAE-graph	7.092 ± 0.015
AE	0.472 ± 0.018
VAE	0.406 ± 0.005
IRAE	2.137 ± 0.023
GeomAE	2.811 ± 0.003
SPAE	0.476 ± 0.013
TopoAE	1.410 ± 0.008
GRAE	0.431 ± 0.005

For the computation of shortest path distances on the graph, we employ the *Dijkstra* algorithm, implemented in the `scipy` Python library. The time complexity of *Dijkstra*'s algorithm is $O(VE + V^2 \log V)$, where V and E represent the number of nodes and edges in the graph, respectively. As an example, for the Swiss Roll dataset with 10k data points and a k-nearest neighbor graph constructed with $k = 10$, resulting in approximately 50k edges, it takes approximately 23.5 seconds to compute the shortest path distances between all nodes using an AMD Ryzen 9 7950X CPU.

One advantage of GGAE is its robustness to errors in global geodesic distance calculations. As demonstrated in Section 5.4, GGAE maintains its performance levels even with only 10% of the original edges. This robustness significantly reduces the time needed to compute shortest path distances on the graph, as shown in Table 4.

Table 4. The elapsed time for shortest path distance calculation on Swiss Roll

disconnecting probability p	Elapsed time (s)
0.00	53.72
0.50	44.49
0.80	34.75
0.85	27.96
0.90	5.08

Furthermore, this robustness allows adjusting the *limit* parameter in the *Dijkstra* algorithm. The *Dijkstra* algorithm stops searching for a path between two nodes if the path distance exceeds the *limit* value, assigning an infinite distance instead. A lower *limit* significantly speeds up the algorithm but at the cost of greater inaccuracy in global distances. However, due to GGAE's robustness to these errors, a low *limit* parameter does not compromise GGAE's performance, while significantly decreasing the computation time for *Dijkstra*. For instance, we use a *limit* of 100 for the Swiss Roll experiments, but reducing the *limit* to 10 decreases the computation time to just 1.69 seconds, without affecting GGAE's superior performance on the Swiss Roll dataset, as shown in Table 5.

Table 5. Performance Comparison of GGAE with different *limit* parameters

limit	recon.(↓)	kNN(↑)	Spear(↑)	$KL_{0.01}$(↓)	$KL_{0.1}$(↓)	KL_1(↓)
10	8.011e-3	0.9910	**1.0000**	4.808e-5	9.551e-6	**2.098e-7**
100	**4.874e-3**	**0.9911**	**1.0000**	**3.673e-5**	**4.730e-6**	5.030e-7

B.2. Evaluation Metrics

In this section, we explain the quantitative metrics of graph geometry preservation, which are *kNN*, *Spear*, and KL_σ ($\sigma = 0.01, 0.1, 1$). These metrics all measure how pairwise distances of data points and pairwise distances of corresponding latent points are aligned. For our purpose of evaluating graph geometry preservation, we use the *semantic distance* (which the data graph $\mathcal{G}$ approximates) as the distance metric for data points, while the distance between latent points are measure by Euclidean distances.

The semantic distance for each dataset is as follows. For *Swiss Roll*, it is the geodesic distance along the surface of 2D Swiss Roll manifold *without hole*. For *dSprites*, it is the 2-norm distance between the latent vectors *(scale, x, y)*. For *Rotating MNIST*, it is only defined for images in a same sequence as the number of time steps between them. Finally for *Robot Images*, it is the 2-norm distance between block positions in images.

B.2.1. *kNN*

The *kNN* (Kobak & Berens, 2019) evaluates the fraction of k-nearest neighbors in the data space that are also a k-nearest neighbors in the latent space. If nearest neighbors are perfectly preserved, *kNN* becomes 1. For the datasets except for Rotating MNIST, we report the average of *kNN* obtained by changing k from 10 to 200 in steps of 10, as proposed in Moor et al. (2020). For Rotating MNIST, we iterate k from 2 to 10 in steps of 2, since a rotating sequence only contains 36 images.

B.2.2. *Spear*

The *Spear* (Kobak & Berens, 2019) is defined as the Spearman's rank correlation between the pairwise distances of data points and the pairwise distances of latent points. This measures how rankings (not the exact distances) of the pairwise distances of data and latent points are aligned. If these are perfectly aligned, *Spear* becomes 1.

B.2.3. KL_σ

The KL_σ measures the Kullback-Leibler divergence between the density estimates (Chazal et al., 2011) computed in the data and latent space. Denote the data and their pairwise distance by $X = \{\mathrm{x}_i\}_{i=1}^N$ and $\mathrm{dist}(\mathrm{x}_i, \mathrm{x}_j)$, and denote latent points by $Z = \{\mathrm{z}_i\}_{i=1}^N$. The density estimate $p_X : X \to \mathbb{R}$ on X is defined by

$$p_{X,\sigma}(\mathrm{x}_i) = \frac{\tilde{p}_{X,\sigma}(\mathrm{x}_i)}{\sum_j \tilde{p}_{X,\sigma}(\mathrm{x}_j)}, \quad \tilde{p}_{X,\sigma}(\mathrm{x}_i) = \sum_j \exp\left(-\frac{1}{\sigma}\left(\frac{\mathrm{dist}(\mathrm{x}_i, \mathrm{x}_j)}{\max_{\mathrm{x}',\mathrm{x}''\in X} \mathrm{dist}(\mathrm{x}', \mathrm{x}'')}\right)^2\right), \tag{25}$$

and the density estimate $p_Z : Z \to \mathbb{R}$ on Z is defined similarly by replacing dist with Euclidean distance. With these, KL_σ is defined by $D_K L(p_{X,\sigma} \| p_{Z,\sigma})$. Larger σ corresponds to evaluating the preservation of geometry on a more global scale. We use $\sigma = 0.01, 0.1, 1$ as in Moor et al. (2020).

B.3. Details of *Dreamer* Architecture

Learning dynamics models in low-dimensional latent state space is an actively studied area, primarily due to its computation- and memory-efficient prediction capabilities (Hafner et al., 2019; 2020a;b). In latent dynamics learning experiments in Section 5.3, we utilize a modified version of the *Dreamer* (Hafner et al., 2019). The original version of the *Dreamer* is proposed for model-based reinforcement learning, consisting of the following components: representation model (encoder), observation model (decoder), transition model (latent dynamics), reward model, value model and action model. In our experiment, we employ models only related to the latent dynamics learning as

$$\text{representation model (encoder):} \quad f_\theta(z_t|x_t) \tag{26}$$
$$\text{observation model (decoder):} \quad g_\theta(x_t|z_t) \tag{27}$$
$$\text{transition model (latent dynamics):} \quad q_\theta(z_t|z_{t-1}). \tag{28}$$

Originally, these three models are trained simultaneously, incorporating both the reconstruction error and the dynamics prediction error into the loss function. To evaluate how helpful the latent representations obtained by GGAE and other baselines are, we keep the pretrained autoencoders fixed and train the dynamics model using only the dynamics prediction loss. The dynamics learning error, as reported in Table 10, is evaluated based on the mean squared error between the future observation image and its prediction.

B.4. Quantitative Evaluation Results

In this section, we present the quantitative evaluation results of each experiment. In all tables except Tables 8 and 11, the best result is indicated by bold and underlined text, while the second-best result is shown in bold. In Tables 8 and 11, only the best result is indicated by bold text.

B.4.1. Swiss Roll

Table 6. Evaluation metrics on the Swiss Roll dataset

	recon.(↓)	kNN(↑)	Spear(↑)	$KL_{0.01}$(↓)	$KL_{0.1}$(↓)	KL_1(↓)
GGAE (ours)	**4.874e-3**	**0.9911**	**1.0000**	**3.673e-5**	**4.730e-6**	**5.030e-7**
SPAE-graph	7.315e-3	0.9784	0.9996	**4.638e-5**	6.201e-5	**8.142e-7**
TopoAE-graph	9.671e-3	**0.9912**	**0.9999**	7.852e-5	**6.804e-6**	8.447e-7
NRAE-graph	**1.768e-3**	0.8026	0.8525	2.166e-1	2.364e-2	6.442e-4
AE	1.134e-1	0.5800	0.5399	1.650e-1	6.037e-2	2.430e-3
VAE	5.903e-2	0.7929	0.4644	1.845e-1	4.268e-2	1.579e-3
IRAE	3.385e-2	0.9447	0.7093	1.093e-2	2.466e-2	1.380e-3
GeomAE	4.130e-2	0.8103	0.8776	1.301e-2	1.433e-2	8.919e-4
SPAE	1.890e-1	0.7049	0.7113	3.636e-2	8.782e-3	7.911e-4
TopoAE	4.448e-2	0.9461	0.9981	5.696e-4	2.769e-4	2.470e-5
GRAE	5.010e-2	0.8353	0.8957	6.902e-2	6.432e-3	3.271e-4

B.4.2. dSprites

Table 7. Evaluation metrics on the dSprites dataset

	recon.(↓)	kNN(↑)	Spear(↑)	$KL_{0.01}$(↓)	$KL_{0.1}$(↓)	KL_1(↓)
GGAE (ours)	2.830e-4	**0.9683**	**0.9979**	**1.633e-3**	**3.310e-4**	**1.291e-5**
SPAE-graph	1.487e-4	**0.8930**	**0.9336**	5.213e-2	1.272e-2	6.152e-4
TopoAE-graph	3.784e-4	0.8415	0.9223	2.845e-2	**9.520e-3**	5.766e-4
NRAE-graph	2.861e-3	0.8068	0.7372	2.216e-1	6.244e-2	2.037e-3
AE	**7.405e-5**	0.8411	0.8717	6.096e-2	2.229e-2	1.155e-3
VAE	**9.597e-5**	0.8160	0.8392	4.945e-2	2.257e-2	1.052e-3
IRAE	2.154e-4	0.8556	0.9292	2.870e-2	1.171e-2	6.576e-4
GeomAE	1.519e-4	0.8437	0.9335	**2.211e-2**	9.630e-3	5.464e-4
SPAE	1.262e-4	0.8550	0.8997	5.421e-2	1.795e-2	8.898e-4
TopoAE	1.987e-4	0.8499	0.9291	2.715e-2	9.588e-3	5.846e-4
GRAE	1.967e-4	0.8281	0.9195	3.208e-2	1.030e-2	**3.954e-4**

SPAE-graph with a low regularization term As mentioned in Section 5.2.2, we encountered challenges in balancing the trade-off between reconstruction and regularization when training SPAE-graph for the dSprites dataset. We report *SPAE-graph(recon.)*, the case with a low reconstruction error, in Figure 3 and Table 7. Here, we present *SPAE-graph(reg.)*, the opposite case with a low regularization term. The latent representation learned by SPAE-graph(reg.) is illustrated in Figure 10 with evaluation metrics provided in Table 8, both in comparison with GGAE and SPAE-graph(recon.). We note that GGAE outperforms SPAE-graph(reg.) even in graph geometry preservation. This is because the shortest path detours the hole in *(scale, x, y)* space, making SPAE-graph preserve exaggerated distances around the hole as in the Swiss Roll experiments.

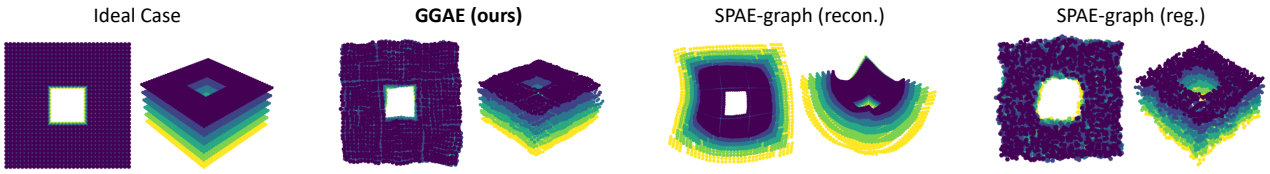

Figure 10. Left: An ideal representation that perfectly preserves the ground truth geometry of the dSprites manifold. *Others*: Three-dimensional latent representations of GGAE, SPAE-graph(recon.) and SPAE-graph(reg.).

Table 8. Evaluation metrics on the dSprites dataset: GGAE vs. SPAE-graph

	recon.(↓)	kNN(↑)	Spear(↑)	$KL_{0.01}$(↓)	$KL_{0.1}$(↓)	KL_1(↓)
GGAE (ours)	2.830e-4	**0.9683**	**0.9979**	**1.633e-3**	**3.310e-4**	**1.291e-5**
SPAE-graph (recon.)	**1.487e-4**	0.8930	0.9336	5.213e-2	1.272e-2	6.152e-4
SPAE-graph (reg.)	4.535e-1	0.9500	0.9941	4.410e-3	7.989e-4	4.116e-5

B.4.3. ROTATING MNIST

Table 9. Evaluation metrics on the Rotating MNIST dataset

	recon.(↓)	kNN(↑)	Spear(↑)	$KL_{0.01}$(↓)	$KL_{0.1}$(↓)	KL_1(↓)
GGAE (ours)	4.640e-2	**1.000**	**0.9984**	**7.048e-4**	**1.432e-5**	**1.434e-7**
SPAE-graph	4.683e-2	**1.000**	**0.9984**	**8.697e-4**	**2.537e-5**	**4.840e-7**
TopoAE-graph	4.070e-2	0.8036	0.3753	1.052e-1	6.519e-2	3.262e-3
NRAE-graph	4.046e-2	0.7442	0.2363	1.921e-1	9.019e-2	3.270e-3
AE	**3.732e-2**	0.7510	0.4016	8.366e-2	4.604e-2	2.797e-3
VAE	3.804e-2	0.7167	0.3364	1.444e-1	7.706e-2	2.638e-3
IRAE	3.860e-2	0.7445	0.4096	4.466e-2	2.920e-2	2.324e-3
GeomAE	3.806e-2	0.7538	0.3437	5.568e-2	2.155e-2	1.884e-3
SPAE	**3.753e-2**	0.7173	0.2567	5.224e-2	4.962e-2	2.817e-3
TopoAE	4.871e-2	0.8425	0.4844	1.353e-2	2.407e-2	2.385e-3
GRAE	3.940e-2	0.7424	0.2980	9.940e-2	6.403e-2	3.250e-3

B.4.4. ROBOT IMAGES

Table 10. Evaluation metrics on the robot image dataset

	recon.(↓)	kNN(↑)	Spear(↑)	$KL_{0.01}$(↓)	$KL_{0.1}$(↓)	KL_1(↓)	dyn.(↓)
GGAE (ours)	3.491e-3	**0.8891**	**0.9946**	**4.939e-3**	**7.268e-4**	**2.778e-5**	**5.927e-3**
SPAE-graph	3.018e-3	**0.9094**	**0.9968**	**1.622e-3**	**4.392e-4**	**1.611e-5**	5.985e-3
TopoAE-graph	3.270e-3	0.7944	0.9126	5.438e-2	8.077e-3	2.401e-4	6.110e-3
NRAE-graph	3.263e-3	0.7774	0.9167	3.197e-2	1.454e-2	4.732e-4	**5.955e-3**
AE	3.297e-3	0.7175	0.8629	4.803e-2	2.083e-2	6.505e-4	8.306e-3
IRAE	**2.831e-3**	0.6941	0.8506	6.221e-2	2.141e-2	6.416e-4	6.084e-3
GeomAE	5.348e-3	0.6096	0.8038	7.999e-2	2.502e-2	6.973e-4	7.107e-3
SPAE	**2.675e-3**	0.6978	0.8042	7.765e-2	2.747e-2	1.017e-3	6.272e-3
TopoAE	3.330e-3	0.7155	0.8742	7.765e-2	1.396e-2	4.019e-4	6.499e-3
GRAE	2.847e-3	0.8021	0.9200	2.900e-2	1.187e-2	4.732e-4	6.181e-3

B.4.5. EXPERIMENTS WITH SPARSE GRAPHS

Table 11. Quantitative evaluation results with sparse graphs

dataset	model	p	recon.	kNN	Spear	$KL_{0.01}$	$KL_{0.1}$	KL_1
Swiss Roll	GGAE	0.5	**8.225e-3**	**0.9901**	**0.9999**	**4.808e-5**	**9.551e-6**	**6.938e-7**
		0.7	**8.568e-3**	**0.9894**	**0.9999**	**3.548e-5**	**2.972e-6**	**3.268e-7**
		0.8	6.921e-3	**0.9871**	**0.9998**	**5.670e-5**	**6.591e-6**	**1.118e-6**
		0.85	9.431e-3	**0.9863**	**0.9998**	**5.615e-5**	**7.973e-6**	**1.480e-6**
		0.9	5.601e-3	**0.9853**	**0.9998**	**6.028e-5**	**9.966e-6**	**1.827e-6**
	SPAE-graph	0.5	**5.109e-3**	0.9731	0.9994	8.296e-5	8.861e-5	1.319e-6
		0.7	**3.459e-3**	0.9690	0.9994	1.859e-4	7.395e-5	1.483e-6
		0.8	4.512e-2	0.9728	0.9995	2.540e-4	6.427e-5	2.154e-6
		0.85	1.234e-1	0.9533	0.9978	2.179e-3	2.732e-4	1.246e-5
		0.9	2.356e-0	0.7772	0.8976	3.232e-2	1.524e-2	1.077e-3
Robot images	GGAE	0.5	3.246e-3	0.7724	0.9955	3.362e-3	5.955e-4	2.004e-5
		0.7	3.300e-3	0.7604	0.9954	3.185e-3	6.218e-4	2.140e-5
		0.8	3.210e-3	**0.7647**	**0.9956**	**2.951e-3**	**5.887e-4**	**2.105e-5**
		0.85	**3.125e-3**	**0.7588**	**0.9956**	**2.697e-3**	**5.990e-4**	**2.090e-5**
		0.9	**3.035e-3**	**0.7767**	**0.9960**	**2.522e-3**	**5.529e-4**	**1.922e-5**
	SPAE-graph	0.5	**2.990e-3**	**0.7803**	**0.9966**	**1.547e-3**	**4.631e-4**	**1.562e-5**
		0.7	**2.924e-3**	**0.7610**	**0.9959**	**1.723e-3**	**5.593e-4**	**1.880e-5**
		0.8	**3.137e-3**	0.7261	0.9932	3.443e-3	9.202e-4	3.064e-5
		0.85	3.583e-3	0.6379	0.9748	1.805e-2	4.423e-3	1.705e-4
		0.9	1.036e-2	0.3144	0.5055	1.531e-1	5.007e-2	1.471e-3

We have conducted an additional analysis about the robustness experiments in Section 5.4, specially the relation between the disconnecting probability and the graph approximation error of the geodesic distances. We calculate the approximation error as MAPE (Mean Absolute Percent Error) between the ground-truth geodesic distance and the shortest path distance on the graph, with the results displayed in Table 12. As the disconnecting probability p increases, the graph becomes sparser, resulting in a higher approximation error. While this leads to performance degradation in SPAE-graph, our GGAE retains its geometry preservation performance, as shown in Figure 11. This result highlights the stability of GGAE against the graph approximation error of the geodesic distance.

Table 12. Approximation errors (MAPE) across various disconnecting probabilities p

	p=0.5	p=0.7	p=0.8	p=0.85	p=0.9
Swiss Roll	6.64%	9.65%	31.90%	63.15%	157.38%
Robot images	3.96%	7.33%	31.28%	62.18%	184.46%

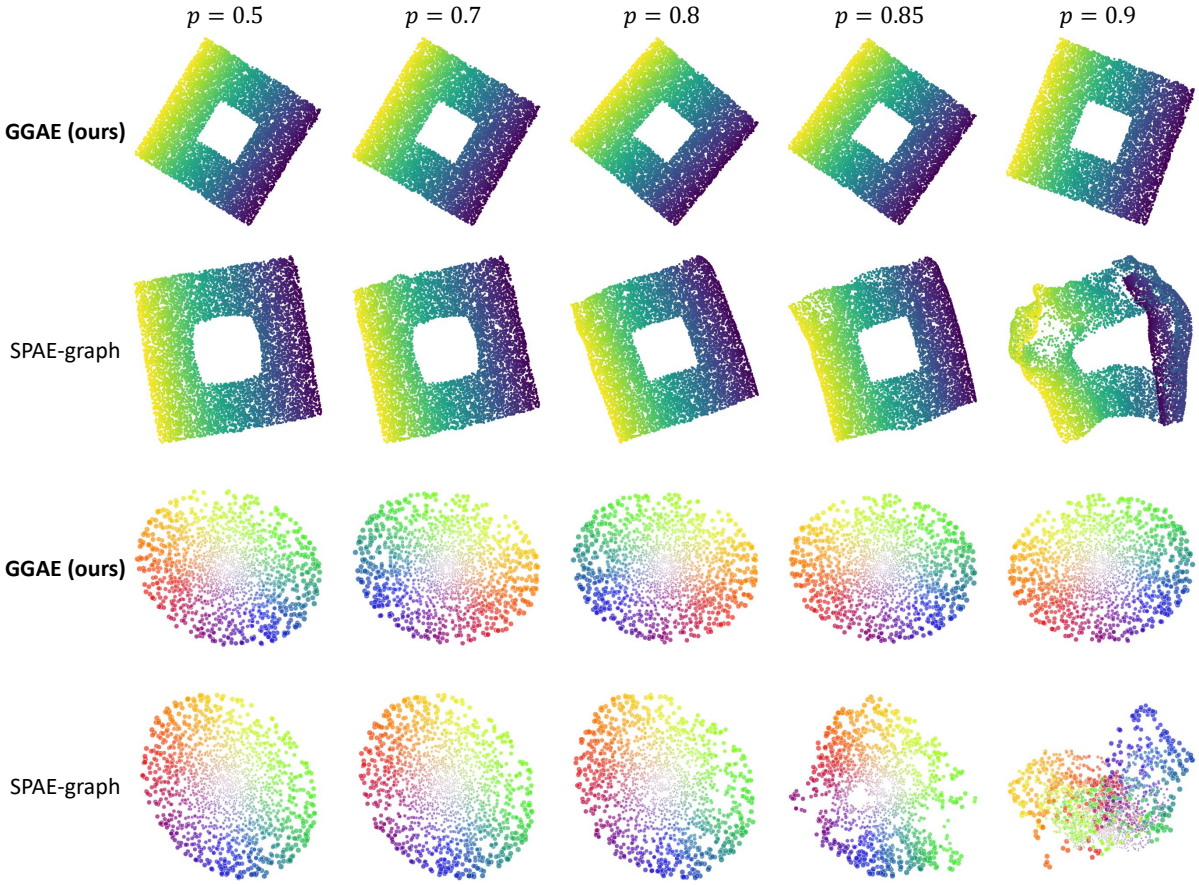

Figure 11. The trained latent representations from GGAE and SPAE-graph across different edge-disconnection probabilities. The top two rows display the trained latent representations of the Swiss Roll dataset, while the bottom two rows show the trained latent representations of the Robot Images dataset, both obtained from GGAE and SPAE-graph.

C. Additional Results on MNIST Dataset

In this section, we conduct additional experiments on MNIST dataset to demonstrate potential application of GGAE in real-world scenarios where ground-truth similarity information is not available. Among the digits in the dataset, 3' and '8' exhibit similar shapes, often resulting in overlapping clusters in the latent space when trained with a vanilla autoencoder, as shown in Figure 12. To separate these two clusters, we construct a similarity graph using class label information. Specifically, we use a k-nearest neighbor graph based on a modified Euclidean distances, where the distance between a pair of images with different labels is increased by a factor of $c > 1$. With our proposed regularization to preserve the similarity graph, GGAE learns latent spaces where '3' and '8' become more clearly separated as the value of c increases, as shown in Figure 12. These results suggest that, with a properly defined graph, GGAE can obtain a latent representation suitable for downstream tasks such as classification and clustering.

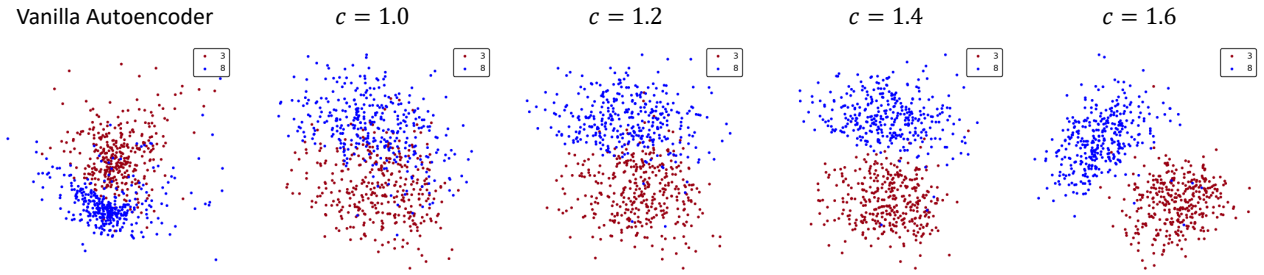

Figure 12. Left: Two-dimensional latent representation learned by vanilla autoencoder. *Others*: Latent representation learned by GGAE, with increasing value of c in graph construction. Larger c results in clearer separation between '3' and '8', as expected.