Supplementary material: Geometrically Enriched Latent Spaces

Georgios Arvanitidis
MPI for Intelligent Systems, Tübingen

Søren Hauberg
DTU Compute, Lyngby

Bernhard Schölkopf
MPI for Intelligent Systems, Tübingen

1 Basics of Riemannian geometry

Let us assume a d-dimensional smooth manifold $\mathcal{M}$ embedded in an ambient space $\mathcal{X} = \mathbb{R}^D$ with $d < D$, where it is defined a Riemannian metric $\mathbf{M}_{\mathcal{X}} : \mathcal{X} \to \mathbb{R}^{D\times D}_{\succ 0}$. Therefore, the space $\mathcal{X}$ is a Riemannian manifold, since $\mathcal{X}$ is a smooth manifold. This directly implies that the simple Euclidean space is a Riemannian manifold as well. Due to the embedding of $\mathcal{M}$ a Riemannian metric is induced in the tangent space $\mathcal{T}_{\mathbf{x}}\mathcal{M}$ by the restriction of the Riemannian ambient metric $\mathbf{M}_{\mathcal{X}}(\cdot)$, even for the simple case $\mathbf{M}_{\mathcal{X}}(\mathbf{x}) = \mathbb{I}_D$. In order to simplify the analysis, we further assume that a global chart map $\phi : \mathcal{M} \to \mathcal{U} \subseteq \mathbb{R}^d$ exists.

Generally, one of the main utilities of a Riemannian manifold $\mathcal{M} \subseteq \mathcal{X}$ is to enable us compute shortest paths therein. Intuitively, the norm $\sqrt{\langle d\mathbf{x}, d\mathbf{x}\rangle_{\mathbf{x}}}$ represents how the infinitesimal displacement vector $d\mathbf{x} \approx \mathbf{x}' - \mathbf{x}$ on $\mathcal{M}$ is locally scaled. Thus, for a curve $\gamma : [0, 1] \to \mathcal{M}$ that connects two points $\mathbf{x} = \gamma(0)$ and $\mathbf{y} = \gamma(1)$, the length on $\mathcal{M}$ or equivalently in $\phi(\mathcal{M})$ using Eq. 2 is measured as

$$\begin{aligned} \textit{length}[\gamma(t)] &= \int_0^1 \sqrt{\langle \dot\gamma(t), \dot\gamma(t)\rangle_{\gamma(t)}}dt \qquad (11)\\ &= \int_0^1 \sqrt{\langle \dot c(t), \mathbf{M}(c(t))\dot c(t)\rangle}dt = \textit{length}[c(t)] \end{aligned}$$

where $\dot\gamma(t) = \partial_t\gamma(t) \in \mathcal{T}_{\gamma(t)}\mathcal{M}$ is the velocity of the curve and accordingly $\dot c(t) \in \mathcal{T}_{c(t)}\phi(\mathcal{M})$. The length is an invariant quantity under reparametrization i.e., for any continuous monotonic function $s : [a, b] \to [0, 1]$ the curve $\tilde\gamma(t') = \gamma(s(t'))$, $t' \in [a, b]$ has the same length. Instead, in order to find the *shortest path* we minimize the corresponding energy functional, which is a non-invariant quantity,

$$\gamma(t)^* = \underset{\gamma(t)}{\operatorname{argmin}} \frac{1}{2}\int_0^1 \langle \dot\gamma(t), \dot\gamma(t)\rangle_{\gamma(t)}dt, \qquad (12)$$

and similarly the energy can be written for the $c(t) \in \phi(\mathcal{M})$ in the intrinsic coordinates as Eq. 11. The minimizers of this energy have constant speed $\|\dot\gamma(t)\|_{\gamma(t)}$and are known as *geodesics*.

In theory, instead of solving the problem on $\mathcal{M}$, we use the intrinsic coordinates. Assuming a global chart $\phi(\cdot)$, we search for a curve $c(t) = \phi(\gamma(t)) \in \phi(\mathcal{M})$ that minimizes the corresponding energy functional $E[c(t)] = \frac{1}{2}\int_0^1 \langle \dot c(t), \mathbf{M}(c(t))\dot c(t)\rangle dt$. Here, we used the fact that $\gamma(t) = \phi^{-1}(c(t)) \Rightarrow \dot\gamma(t) = \mathbf{J}_{\phi^{-1}}(c(t))\dot c(t)$, since by the definition of a smooth manifold $\phi(\cdot)$, $\phi^{-1}(\cdot)$ exist and are smooth maps. Now, we are able to find the minimizers by directly applying the Euler-Lagrange equations to the energy $E[c(t)]$, which results to a system of 2^{nd} order non-linear ordinary differential equations (ODEs) written as in Arvanitidis et al. (2018)

$$\begin{aligned} \ddot c(t) = -\frac{1}{2}\mathbf{M}^{-1}(c(t))\Big[2(\dot c(t)^\intercal \otimes \mathbb{I}_d)\frac{\partial \text{vec}[\mathbf{M}(c(t))]}{\partial c(t)}\dot c(t)\\ - \frac{\partial \text{vec}[\mathbf{M}(c(t))]}{\partial c(t)}^\intercal (\dot c(t) \otimes \dot c(t))\Big], \qquad (13) \end{aligned}$$

where vec$[\cdot]$ stacks the columns of a matrix into a vector and $\otimes$ is the Kronecker product. This is solved as a boundary value problem (BVP) with boundary conditions $c(0) = \mathbf{x}$ and $c(1) = \mathbf{y}$. Note that this ODEs system is a standard result in differential geometry, and intuitively, the resulting shortest paths tend to avoid areas with high metric magnitude $\sqrt{|\mathbf{M}(c(t))|}$.

We can do computations on $\mathcal{M}$, or equivalently in $\phi(\mathcal{M})$, using two operations analogous to the "plus" and "minus" of the Euclidean space. First, the *exponential map* is an operator $\text{Exp}_{\mathbf{x}}(\mathbf{v}t) = \gamma(t)$ that takes two inputs, a point $\mathbf{x} \in \mathcal{M}$ and a $\mathbf{v} \in \mathcal{T}_{\mathbf{x}}\mathcal{M}$, and generates a geodesic with $\gamma(1) = \mathbf{y} \in \mathcal{M}$ and initial velocity $\dot\gamma(0) = \mathbf{v}$. The inverse operator is called the *logarithmic map* $\text{Log}_{\mathbf{x}}(\mathbf{y}) = \mathbf{v}$ that takes two inputs $\mathbf{x}, \mathbf{y} \in \mathcal{M}$ to return the tangent vector $\mathbf{v} \in \mathcal{T}_{\mathbf{x}}\mathcal{M}$. Note that these two operators are dual in a small neighborhood around $\mathbf{x} \in \mathcal{M}$. Moreover, the logarithmic map provides *coordinates* for the points in a neighborhood on $\mathcal{M}$ with respect to the base point $\mathbf{x}$, but only the distances between the center $\mathbf{x}$ and the points are meaningful and not the ones between the points. Also, by definition the $\langle \text{Log}_{\mathbf{x}}(\mathbf{y}), \mathbf{M}_{\mathcal{X}}(\mathbf{x})\text{Log}_{\mathbf{x}}(\mathbf{y})\rangle = \text{dist}^2(\mathbf{x}, \mathbf{y}) = \text{length}^2[\gamma(t)]$, but we can rescale the logarithmic map such that the $\text{dist}^2(\mathbf{x}, \mathbf{y}) = \langle \overline{\text{Log}}_{\mathbf{x}}(\mathbf{y}), \overline{\text{Log}}_{\mathbf{x}}(\mathbf{y})\rangle$. The rescaled coordinates $\overline{\text{Log}}_{\mathbf{x}}(\mathbf{y})$ are known as *normal coordinates* and are the ones that we use in practice.

2 Theoretical analysis of the generator

In this section we analyze the properties that the proposed generator $g(\cdot)$ should have. In particular, in order to have a theoretically sound model the generator has to be at least twice differentiable, and additionally, an immersion. Also, we need a specific behavior such that to properly capture the structure of the data manifold in the latent space, both in the stochastic and deterministic generator case. Of course, the basic assumption is that the data lie *near* an *embedded* smooth manifold $\mathcal{M}$ in the ambient space $\mathcal{X}$. Intuitively, an embedded d-dimensional manifold can be considered as a surface that is everywhere homeomorphich to a d-dimensional Euclidean space, which implies that contractions and intersections are now allowed. In contrast, an *immersion* is a relative simpler condition, since intersections are allowed but again no contractions. In theory, the generator has to be at least an immersion such that to compute the pull-back Riemannian metric of the manifold.

Stochastic generator. We consider as generator the function $g(\mathbf{z}) = \mu(\mathbf{z}) + \text{diag}(\varepsilon) \cdot \sigma(\mathbf{z})$, where $\varepsilon \sim \mathcal{N}(\mathbf{0},\ \mathbb{I}_D)$ and $\mu : \mathcal{Z} \to \mathcal{X}$ is a DNN and $\sigma : \mathcal{Z} \to \mathbb{R}_{>0}^{\dim(\mathcal{X})}$ is based on a positive RBF. Note that in principle we model the precision $\beta(\mathbf{z}) = (\sigma^2(\mathbf{z}))^{-1}$ with the positive RBF, so the $\sigma(\mathbf{z}) = \beta^{-1/2}(\mathbf{z})$. From the theory we know that $g(\cdot)$ has to be smooth. At first, we can achieve smoothness easily for $\mu(\cdot)$ and $\sigma(\cdot)$. In particular, $\sigma(\cdot)$ is smooth as a linear combination of smooth functions. For the DNN $\mu(\cdot)$ we can use smooth activation functions as the $\texttt{tanh}(\cdot)$, $\texttt{softplus}(\cdot)$, etc. But the stochasticity of ε makes $g(\cdot)$ non-smooth, and hence, non differentiable with respect to $\mathbf{z}$. Instead, if ε is fixed $\forall\ \mathbf{z} \in \mathcal{Z}$ denoted as $g_\varepsilon(\cdot)$, then this is a smooth nonlinear map, and consequently, differentiable. A different perspective on the smoothness of $g(\cdot)$ has been given by Eklund and Hauberg (2019). There it is shown that $g_\varepsilon(\cdot)$ is actually the random projection of the deterministic smooth nonlinear map $\mathbf{z} \mapsto [\mu(\mathbf{z}),\ \sigma(\mathbf{z})]$ under the random projection matrix $\mathbf{P}_\varepsilon = [\mathbb{I}_D,\ \text{diag}(\varepsilon)]$. In both views, fixing ε implies that the *sampled* $\mathcal{M}_\varepsilon = g_\varepsilon(\mathcal{Z})$ is a smooth immersed manifold in $\mathcal{X}$. Obviously, the $\mathbb{E}_\varepsilon[\mathcal{M}_\varepsilon] = \mathbb{E}_\varepsilon[g_\varepsilon(\mathcal{Z})] = \mathbb{E}_\varepsilon[g(\mathcal{Z})] = \mu(\mathcal{Z})$, which shows that the expected manifold, as well as the likelihood of the individual points $p(\mathbf{x}|\mathbf{z})$ do not change.

The $g_\varepsilon(\cdot)$ is an immersion if $\mathbf{J}_{g_\varepsilon}(\mathbf{z}) = \mathbf{J}_\mu(\mathbf{z}) + \text{diag}(\varepsilon) \cdot \mathbf{J}_\sigma(\mathbf{z})$ is full rank $\forall\ \mathbf{z} \in \mathcal{Z}$. For $\mu(\cdot)$ (DNN) this can be true within the support of $p(\mathbf{z})$ where the activation functions typically do not reach their limit behavior. For instance, with $\texttt{tanh}(\cdot)$ as activation, we expect within the support of $p(\mathbf{z})$ the hidden units output to not be constant ± 1. In addition, we need each hidden layer to have greater or equal number of units to the previous layer while all the weight matrices to be full rank. In this way, we avoid *contractions* of the immersed space. While for $\sigma(\cdot)$ (inverse positive RBF) at least $\dim(\mathcal{Z})$ basis functions has to be active, which holds in general since RBF has infinite support, and the weight matrix has to be full rank. The conventions above define an immersed manifold $\mathcal{M}_\varepsilon = g_\varepsilon(\mathcal{Z})$ in $\mathcal{X}$. Of course, the two Jacobian matrices should not cancel columns. Actually, the immersion is necessary such that the (expected) Riemannian metric $\mathbf{M}(\cdot) = \mathbf{J}_\mu(\cdot)^\intercal \mathbf{J}_\mu(\cdot) + \mathbf{J}_\sigma(\cdot)^\intercal \mathbf{J}_\sigma(\cdot)$ to be full rank.

Generator with linear extrapolation. We analyze the behavior of the proposed architecture $g(\mathbf{z}) = f(\mathbf{z}) + \mathbf{U} \cdot \text{diag}([\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_d}]) \cdot \mathbf{z} + \mathbf{b}$ where $f : \mathcal{Z} \to \mathcal{X}$ is a nonlinear map and $\mathcal{Z} = \mathbb{R}^d$, $\mathcal{X} = \mathbb{R}^D$ with $D \gg d$. Note that for the stochastic generator case and for fixed ε the $f_\varepsilon(\mathbf{z}) = \mu(\mathbf{z}) + \text{diag}(\varepsilon) \cdot \sigma(\mathbf{z})$ is the addition of two nonlinear functions (see above). The linear map is constructed using the top d-eigenvectors scaled by their eigenvalues, coming from the eigendecomposition of the data empirical covariance matrix $\mathbf{C} = \frac{1}{N-1} \sum_{n=1}^{N} (\mathbf{x}_n - \mathbf{b})(\mathbf{x}_n - \mathbf{b})^\intercal$, where $\mathbf{b} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$, which can be decomposed as $\mathbf{C} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\intercal$. We use for $\mathbf{U}$ the first d columns of $\mathbf{V} \in \mathbb{R}^{D \times D}$ and the corresponding eigenvalues. We study if and when $g(\cdot)$ satisfies the following properties:

Smoothness. We need the generator to be sufficiently smooth, which means in our case at least twice differentiable. This condition can be easily satisfied by selecting the activation functions accordingly as $\texttt{tanh}(\cdot), \texttt{softplus}(\cdot)$, etc. In practice, this is necessary since in the geodesic ODEs system we need to compute the derivative of the metric tensor, which in our case is implemented by first taking the derivative of the Jacobian $\mathbf{J}_g(\cdot)$. Obviously, by including in $g(\cdot)$ the linear map $\mathbf{A} = \mathbf{U} \cdot \text{diag}([\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_d}])$ and $\mathbf{b}$, the smoothness property will not change.

Immersion. In theory a mapping $g : \mathbb{R}^d \to \mathbb{R}^D$ with $D \gg d$ is an immersion if the corresponding $\mathbf{J}_g(\mathbf{z}) \in \mathbb{R}^{D \times d}$ is everywhere injective or in other words full rank. In our case, the Jacobian includes a neural network and for an example we consider the simple function $f(\mathbf{z}) = \mathbf{W}_1 \cdot \psi(\mathbf{W}_0 \mathbf{z} + \mathbf{b}_0) + \mathbf{b}_1$ with $\psi(\cdot)$ the activation function, and thus, the Jacobian is $\mathbf{J}_f(\mathbf{z}) = \mathbf{W}_1 \cdot \psi'(\mathbf{W}_0 \mathbf{z} + \mathbf{b}_0) \odot \mathbf{W}_0$. In order to be this quantity an immersion, first we need each hidden layer to have more or equal number of hidden units from the previous layer and the weight matrices to have full rank. Additionally, since the derivative of the activation functions $\psi'(\cdot)$ appears, we need this to be non-zero. Otherwise, this will directly affect the total

rank of the Jacobian, because it will reduce the rank of the corresponding weight matrix or simply make it zero. We *conjecture* that for generative models which are trained using a compact support prior $p(\mathbf{z})$ like the Gaussian, the trained model uses the activation functions $\psi(\cdot)$ closer to the center of their domain, where their corresponding derivative $\psi'(\mathbf{z})$ is not zero, and not towards the domain limits. This basically implies that the corresponding hidden unit is active and is used by the model. While if the weight matrices do not have this specific "pyramidal" structure, then we cannot guarantee a full rank Jacobian, since in general holds that $\text{rank}(\mathbf{XY}) \leq \min[\text{rank}(\mathbf{X}), \text{rank}(\mathbf{Y})]$ for any two matrices $\mathbf{X},\ \mathbf{Y}$.

In our case, the Jacobian is $\mathbf{J}_g(\mathbf{z}) = \mathbf{J}_f(\mathbf{z}) + \mathbf{A}$, which means that in theory there are cases where the two matrices could cancel some of their columns. This will directly break the full rank condition, and thus, the mapping at this point will not be an immersion. Practically, this means that the corresponding Riemannian metric tensor in $\mathcal{Z}$, computed as the $\mathbf{J}_g(\mathbf{z})^\intercal \mathbf{J}_g(\mathbf{z})$, will be degenerate since it will not have full rank. However, even if in theory this is a case that could happen, in practice, we *conjecture* that this is a relatively unrealistic scenario. Instead, if the $\mathbf{J}_f(\mathbf{z})$ has low rank the linear part $\mathbf{A}$ could even fix the problem, of course, if any of the rest columns do not cancel each other.

In practice, the weights of $g(\cdot)$ are initialized randomly. Empirically, we always observed that the generator is an immersion. Also, we expect that it is unlikely the rank of the weight matrices to reduce, since we do not regularize them in this direction. In addition, the linear map and the uncertainty term when present, help the Riemannian metric to be full rank. However, in the general setting, a collection of "training tricks" are available to ensure full rank, such as using manifold optimization to ensure that weight matrices lie on the manifold of full rank matrices, regularization schemes that penalize low-rank weight matrices and post-hoc smoothing of the weight matrices.

Extrapolation. The proposed meaningful extrapolation can be used for deterministic generator together with an ambient metric in order to properly capture in $\mathcal{Z}$ the structure or topology of the data manifold. Especially, the behavior of the ambient metric is to be small only near the given data, which pulls the shortest paths towards the data manifold. Similarly, in the stochastic generator case meaningful uncertainty quantification is utilized in order to properly capture in $\mathcal{Z}$ the structure of the data manifold or in some sense its topology (Hauberg, 2018).

Thus, let us consider the deterministic generator case where $g(\mathbf{z})$ is simply a neural network $f(\mathbf{z})$ and let us pick a direction $\mathbf{z}$ so that we move on the line $t\mathbf{z}$ for $t \in \mathbb{R}$. When the $\texttt{tanh}(\cdot)$ activation function is used, as we move further from the support of the prior, the units of the first hidden layer will tend to output always a constant value $\to +1$ or $\to -1$. This means that the extrapolation will not be meaningful since it is gonna be always a constant. Similarly, for the $\texttt{softplus}(\cdot)$ as we move to the boundaries of the domain of t, the output of the activation will be either a constant $\to 0$ or a linear function. However, for each output dimension $f_j(\mathbf{z}t)$ if the $t \to +\infty$ corresponds to a linear extrapolation the $t \to -\infty$ will extrapolate to zero. Therefore, in the $\texttt{softplus}(\cdot)$ case, even if the generator will potentially extrapolate meaningfully in some parts, in general, the behavior is arbitrary and hard to interpret $\forall\ \mathbf{z} \in \mathcal{Z}$. We show the behavior on a synthetic example in Fig. 11.

So the linear map $\mathbf{A}, \mathbf{b}$ could potentially fix the extrapolation issue, since the map $g(\cdot)$ after some threshold t becomes linear. However, as regards the immersion condition, when $t \to \pm\infty$ if all the dimension $f_j(\mathbf{z}t)$ cancel out the corresponding rows of $\mathbf{A}\mathbf{z}t + \mathbf{b}$, then the $g(t\mathbf{z})$ output will be a constant value. However, we argue again that this is quite unrealistic to happen on the same time for all the output D dimensions.

Above, we only describe the theoretical conditions and the properties that a generator has to respect. Nevertheless, proper guarantees and analysis should be provided in the future.

Linear projection of the ambient space. Additionally, we discuss the case where we linearly project the data manifold in $\mathcal{X}' = \mathbb{R}^{d'}$, a lower dimensional space $D > d' > d$, and we learn the ambient metric $\mathbf{M}_{\mathcal{X}'}(\cdot)$ therein. Intuitively, instead of finding the shortest path $\gamma(t)$ on the $\mathcal{M} \subset \mathcal{X}$ we find the path on the projected manifold in $\mathcal{X}'$ and we expect that the actual structure of $\mathcal{M}$ is preserved. The reason of this step is to remove the non very informative extra dimensions from the data e.g. high frequency context, which do not provide any significant information regarding the structure of that data manifold or simply if they just correspond to noise. In other words, this step helps us to reduce the dimensionality, such that to construct the "ambient" metric using the projected data. Of course, this is only acceptable if the linear projection does not change the structure of the data manifold, for instance by introducing self intersections or contractions. Note that still the generator is trained between the space $\mathcal{Z}$ and $\mathcal{X}$, prior to the linear projection, so $g(\cdot)$ is still able to capture the high frequency context of the given data.

The practical reason for this step is that for high dimensional data as images, we need to reduce the di-

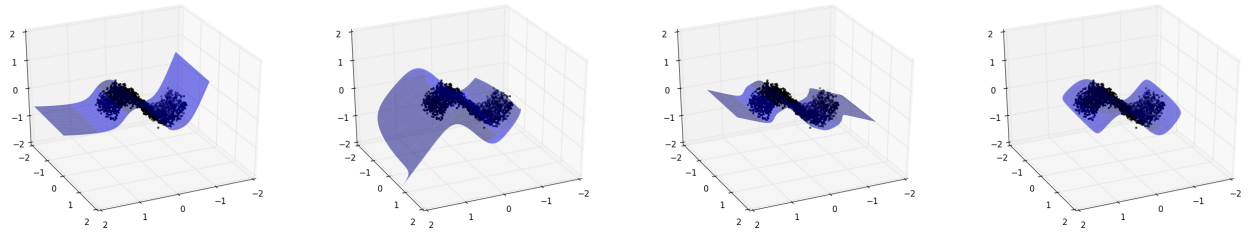


Figure 11: We trained an MLP with 2 hidden layers from $g : \mathbb{R}^2 \to \mathbb{R}^3$, with hidden layer sizes 2, 3. From *left* to *right*: The extrapolation result with activation function `softplus`$(\cdot)$ and including the proposed linear map, without the linear map, with activation function `tanh`$(\cdot)$ and including the proposed linear map, without the linear map. We see that the linear map improves the extrapolation of $g(\cdot)$, and the change is faster along the direction with the largest eigenvalue.

mension of $\mathcal{X}$ due the curse of dimensionality (Bishop, 2006). Especially, if the learned ambient metric is based on pairwise Euclidean distances. Also, we know that, even locally, Euclidean distance makes not too much sense for images. Hence, the linear projection to a lower dimensional space $\mathcal{X}'$ helps us to ensure that at least locally straight lines will be more meaningful.

Therefore, the linear projection of the data, helps us to learn easier an "ambient" Riemannian metric that provides information regarding the structure of the actual data manifold. However, we note again that it is necessary this step to not change the structure of the data manifold. Thus, the metric $\mathbf{M}_{\mathcal{X}'}(\cdot)$ that is learned from the projected data is defined in $\mathbb{R}^{d'}$, and hence, the pull-back in the latent space becomes

$$\begin{aligned}\langle \mathbf{v}', \mathbf{M}_{\mathcal{X}'}(\mathbf{x}')\mathbf{v}' \rangle &= \langle \mathbf{P}\overline{\mathbf{v}}, \mathbf{M}_{\mathcal{X}'}(\mathbf{P}(\overline{\mathbf{x}} - \overline{\mathbf{c}}))\mathbf{P}\overline{\mathbf{v}} \rangle \\ &= \langle \mathbf{v}, \mathbf{J}_g(\mathbf{z})^\intercal \mathbf{P}^\intercal \mathbf{M}_{\mathcal{X}'}(\mathbf{P}(g(\mathbf{z}) - \overline{\mathbf{c}}))\mathbf{P}\mathbf{J}_g(\mathbf{z})\mathbf{v} \rangle, \end{aligned} \quad (14)$$

where $\mathbf{v}', \mathbf{x}' \in \mathbb{R}^{d'}$ the point and the tangent vector in $\mathcal{X}'$, $\mathbf{P} \in \mathbb{R}^{d' \times D}$ is the projection matrix derived from PCA with $\overline{\mathbf{c}} \in \mathbb{R}^D$ the center of the data, $\overline{\mathbf{x}}$, $\overline{\mathbf{v}} \in \mathbb{R}^D$ the point and the tangent vector in $\mathcal{X}$ and $\mathbf{z}, \mathbf{v} \in \mathbb{R}^d$ the latent space inputs with the Jacobian $\mathbf{J}_g(\mathbf{z}) \in \mathbb{R}^{D \times d}$. Note that we can directly use the same setting when $g(\cdot)$ is a stochastic generator.

A simple constructive example is to consider the data in Fig. 11, and expand the dimensions by concatenating 100 columns with noise sampled from $\varepsilon_i \sim \mathcal{N}(0, 0.001^2)$ as $[\mathbf{x}, \varepsilon_1, \ldots, \varepsilon_{100}]$. The generator is trained as $g : \mathcal{Z} \to \mathcal{X}$. Obviously, the structure of the actual data manifold will not be different in $\mathcal{X} = \mathbb{R}^{103}$, and also, we can "project" it in $\mathcal{X}' = \mathbb{R}^3$ by excluding the last 100 columns. Therefore, we can construct the "ambient" metric $\mathbf{M}_{\mathcal{X}'}(\cdot)$ in $\mathcal{X}'$, where the structure of the data manifold is preserved. Thus, a shortest path in $\mathcal{Z}$ will move optimally on $\mathcal{M} \subset \mathcal{X}$, while respecting its geometry, and in such a way that after the projection step, the resulting path will respect in $\mathcal{X}'$ the geometry that is represented by the $\mathbf{M}_{\mathcal{X}'}(\cdot)$.

As regards the real data, the extra dimensions might not be noise, but high frequency context, which does not affect the underlying structure of the manifold. Hence, the shortest paths computed in $\mathcal{X}'$ are able to approximate closely the true paths on the actual data manifold $\mathcal{M} \subset \mathcal{X}$, as long as the linear projection step does not change the structure of $\mathcal{M}$ in $\mathcal{X}'$.

3 Details for the construction of ambient Riemannian metrics

In this section we provide the details for constructing the metrics that have been used in the paper. As we discussed above, the ambient metrics can be either constructed in $\mathcal{X}$ or in lower dimensional space $\mathcal{X}'$ where we project linearly the given data manifold.

3.1 Local linear discriminant analysis based Riemannian metric

To compute the ambient metric for a test point $\mathbf{x} \in \mathcal{X} = \mathbb{R}^D$ using the local LDA we have first to learn the base metrics for a set of points $\mathcal{S} = \{\mathbf{x}_s\}_{s=1}^S$ following the approach of Hastie and Tibshirani (1994), and then, compute the weighted average (see Sec 3.1, Eq. 4). Based on a given labeled set $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$ the metric at each $\mathbf{x}_s$ is defined as

$$\mathbf{M}_s = \mathbf{W}_s^{-1} \mathbf{B}_s \mathbf{W}_s^{-1} + \varepsilon \mathbf{W}_s^{-1}, \quad (15)$$

where $\epsilon > 0$ a small scalar to avoid degenerate metrics, the $\mathbf{W}_s \in \mathbb{R}^{D \times D}$ is called the within covariance matrix and $\mathbf{B}_s \in \mathbb{R}^{D \times D}$ the in-between covariance matrix. Let K be the number of the k-nearest neighbors denoted with the set $\text{knn}(\mathbf{x}_s)$ computed under the initial $\mathbf{M}_s = \mathbb{I}_D$ and $d_s = \left\| \mathbf{M}_s^{1/2}(\mathbf{x} - \mathbf{x}_s) \right\|_2$. We use a weighting function $w_s(\mathbf{x}) = \left[1 - (d_s/\sigma_s)^3\right]^3 \cdot 1_{\{d_s < \sigma_s\}}$ where $\sigma_s = \max_{k \in \text{knn}(\mathbf{x}_s)} d_k$. Then, from the labeled point set we consider only the ones that are within the

$\text{knn}(\mathbf{x}_s)$, so the within and in-between matrices are

$$\mathbf{W}_s = \frac{\sum_{c\in\mathcal{C}} \sum_{n:y_n=c} w_s(\mathbf{x}_n)(\mathbf{x}_n - \mathbf{m}_c)(\mathbf{x}_n - \mathbf{m}_c)^\intercal}{\sum_k^K w_s(\mathbf{x}_k)}, \tag{16}$$

$$\mathbf{m}_c = \frac{1}{\sum_{k:y_k=c} w_s(\mathbf{x}_k)} \sum_{n:y_n=c} w_s(\mathbf{x}_n)\mathbf{x}_n, \tag{17}$$

$$\mathbf{B} = \sum_{c\in\mathcal{C}} \pi_c(\mathbf{m}_c - \mathbf{m})(\mathbf{m}_c - \mathbf{m})^\intercal, \tag{18}$$

$$\pi_c = \frac{\sum_{n:y_n=c} w_s(\mathbf{x}_n)}{\sum_{k=1}^K w_s(\mathbf{x}_k)}. \tag{19}$$

Using the updated metrics $\mathbf{M}_s$ we iterate the procedure i.e. finding the $\text{knn}(\mathbf{x}_s)$, computing d_s, etc, until either a fixed point is found i.e., the $\mathbf{M}_s$ matrices do not change, or if we exceed a pre-specified number of iterations. Moreover, we use only the diagonal $\mathbf{W}_s$ since in higher dimensions is easier to get degenerate metrics when this matrix is full.

As we discussed in the main paper the construction of the base metrics $\mathbf{M}_k$ is problem dependent. Hence, these can be constructed in any meaningful way, such that to provide the essential high-level information or domain knowledge for the problem we want to model. For further examples, we could construct these metrics based on ordinal information between points or even triplet constraints. In general, this is a metric learning problem Suárez et al. (2018) and we construct the ambient metric depending on the problem of interest.

3.2 Density and data support based Riemannian metric

In order to construct a probability density function based ambient metric, essentially, we want to roughly estimate the density of the high dimensional data. A relatively simple, easy and robust model to learn such a density is the Gaussian Mixture Model (GMM). So in practice, we want to learn a $h(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{\mu}_k,\ \boldsymbol{\Sigma}_k)$, with $\sum_{k=1}^K \pi_k = 1$. However, we have to pay attention to some details. First we want to avoid centers $\boldsymbol{\mu}_k$ with huge covariance $\boldsymbol{\Sigma}_k$ that are placed outside of the data distribution. For that reason we chose to use the same covariance matrix for all the data $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$. Intuitively, we want this covariance to be roughly a spherical one, in order to cover the whole data manifold with balls or ellipsoids. So we chose $\boldsymbol{\Sigma} = \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_D^2)$.

A second problem is that in high dimensions if $\sigma_d < 1$, then the $|\boldsymbol{\Sigma}|$ converges to zero, and consequently, the normalization constant. For this reason we use the unnormalized Gaussian mixture model and this is not a problem, because all of the components share the same covariance. Of course, we are still able to set the parameters α, ε such that to lower and upper bound the metric. In some sense, these parameters define one aspect of the manifold's curvature, since they define how big is the difference of the metric between the points where $h(\mathbf{x}) \to 0$ and $h(\mathbf{x}) \to 1$.

One drawback of this method, is that the metric will shrink the distances accordingly to the data density in the ambient space i.e. as the density becomes higher the distance shrinks. Obviously, in some cases this might be a meaningful behavior. However, we might want to simply move near the data and not necessarily analogous to the corresponding density. So a close related approach is to utilize a positive function $h(\mathbf{x}) = \sum_k w_k \phi_k(\mathbf{x})$, with $w_k > 0$ and $\phi_k(\mathbf{x}) = \exp(-0.5 \cdot \lambda_k \left\|\mathbf{x} - \mathbf{c}_k\right\|_2^2)$, that is trained in such a way that the output near the given data is $h(\mathbf{x}) \to 1$, otherwise $h(\mathbf{x}) \to 0$. One way to train the parameters is to fix $\mathbf{c}_k$ using k-means, setting the bandwidth $\lambda_k = \frac{1}{2}\left[\kappa \frac{1}{|\mathcal{C}_k|} \sum_{\mathbf{x}\in\mathcal{C}_k} \left\|\mathbf{x} - \mathbf{c}_k\right\|_2\right]^{-2}$ where $\kappa > 0$ a scaling factor, $\mathcal{C}_k$ the points in the cluster of $\mathbf{c}_k$, and the w_k can be found using a closed form solution or gradient descent under the mean squared error $L(\mathbf{w}) = \sum_{n=1}^N \left\|1 - h(\mathbf{x}_n)\right\|_2^2$. Obviously, this is a relatively simple model, however, it models very well the desired behavior of the ambient metric.

3.3 Cost based Riemannian metric

The cost related ambient Riemannian metric essentially pulls the shortest paths towards regions of the ambient space $\mathcal{X}$ with low cost. In our experiments we used a relatively simple and interpretable cost function utilizing again the RBF network $h(x) = \sum_k y_k \phi_k(\mathbf{x})$ with basis functions $\phi_k(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x} - \mathbf{c}_k\right\|_2^2\right)$ and $y_k > 0$ some given values. Apart from the simplicity, this type of cost function has a very interpretable behavior, since it defines regions in $\mathcal{X}$ where the cost is high and the corresponding regions in $\mathcal{Z}$ will be avoided by the shortest paths. Intuitively, these can be neighborhoods of points in $\mathcal{X}$ that we want to avoid as we move on the data manifold.

3.4 Can we construct the $\mathbf{M}_{\mathcal{X}}(\cdot)$ in $\mathcal{Z}$?

A logical question is, why we do not construct the informative metric directly in $\mathcal{Z}$ using the latent codes, and simply, combine it linearly with the pull-back metric that is induced by the generator? The answer is quite straight forward though. The metric $\mathbf{M}_{\mathcal{X}}(\cdot)$ is mainly based on Euclidean distances. Therefore, the definition of this Riemannian metric in the latent space $\mathcal{Z}$ is impossible, since using the Euclidean distance in $\mathcal{Z}$ is fundamentally wrong and misleading.

4 Expected Riemannian metric approximation in latent space

Here we discuss the approximation to the true expected Riemannian metric, where we evaluate the ambient metric only on the expected generated $\mathcal{M}_{\mathcal{Z}} = \mu(\mathcal{Z})$. In particular, the true stochastic Riemannian metric in the latent space is written as

$$\begin{aligned}\mathbf{M}_{\varepsilon}(\mathbf{z}) &= [\mathbf{J}_{\mu}(\mathbf{z}) + \text{diag}(\varepsilon)\cdot\mathbf{J}_{\sigma}(\mathbf{z})]^{\intercal} \\ &\mathbf{M}_{\mathcal{X}}\left(\mu(\mathbf{z}) + \text{diag}(\varepsilon)\cdot\sigma(\mathbf{z})\right)[\mathbf{J}_{\mu}(\mathbf{z}) + \text{diag}(\varepsilon)\cdot\mathbf{J}_{\sigma}(\mathbf{z})],\end{aligned} \tag{20}$$

for which we can approximate the expectation in the latent space as $\mathbf{M}(\mathbf{z}) = \mathbb{E}_{\varepsilon\sim p(\varepsilon)}[\mathbf{M}_{\varepsilon}(\mathbf{z})]$ with $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbb{I}_D)$. Even if this is a doable computation, in practice, we need to estimate this metric, as well as, its derivative for all the computations on a Riemannian manifold. This directly means that the computational cost will be extremely high, and hence, prohibited. For this reason we provide the following relaxation

$$\begin{aligned}\mathbf{M}(\mathbf{z}) = &\mathbf{J}_{\mu}(\mathbf{z})^{\intercal}\mathbf{M}_{\mathcal{X}}\left(\mu(\mathbf{z})\right)\mathbf{J}_{\mu}(\mathbf{z}) \\ &+\mathbf{J}_{\sigma}(\mathbf{z})^{\intercal}\mathbf{M}_{\mathcal{X}}\left(\mu(\mathbf{z})\right)\mathbf{J}_{\sigma}(\mathbf{z}).\end{aligned} \tag{21}$$

Here we are based on the realistic assumption that the generator's uncertainty in the regions of the latent space with representations of the training data to be $\sigma(\mathbf{z}) \to \mathbf{0}$. The reason is that $\mu(\mathbf{z})$ is trained to reconstruct sufficiently well the training data $\mathbf{x}$, and we are also based on the main assumption that the training data lie *near* a manifold $\mathcal{M} \subset \mathcal{X}$. This essentially implies that the corresponding deviation of $\mathbf{x}$ from $\mathcal{M}$ will be negligible and our $g(\mathcal{Z}) = \mathcal{M}_{\mathcal{Z}} \approx \mathcal{M}$. Therefore, the Eq. 20 becomes first

$$\begin{aligned}\widetilde{\mathbf{M}}_{\varepsilon}(\mathbf{z}) = &[\mathbf{J}_{\mu}(\mathbf{z}) + \text{diag}(\varepsilon)\cdot\mathbf{J}_{\sigma}(\mathbf{z})]^{\intercal} \\ &\mathbf{M}_{\mathcal{X}}\left(\mu(\mathbf{z})\right)[\mathbf{J}_{\mu}(\mathbf{z}) + \text{diag}(\varepsilon)\cdot\mathbf{J}_{\sigma}(\mathbf{z})],\end{aligned} \tag{22}$$

and wel compute this expectation to get the $\mathbf{M}(\mathbf{z}) = \mathbb{E}_{\varepsilon}[\widetilde{\mathbf{M}}_{\varepsilon}(\mathbf{z})]$ that is shown in Eq. 21. As regards the regions far from the latent codes where $\sigma(\mathbf{z}) \gg 0$, the $\mathbf{J}_{\sigma}(\cdot)$ will be the dominant term, and hence, the contribution of $\mathbf{M}_{\mathcal{X}}(\mu(\mathbf{z}))$ or even $\mathbf{M}_{\mathcal{X}}(\mu(\mathbf{z}) + \text{diag}(\varepsilon)\cdot\sigma(\mathbf{z}))$ will be negligible there anyways.

In order to demonstrate this behavior, we generate a dataset near $\mathcal{M}$ as $\mathbf{x} = [x_1, x_2, \sin(x_1)]$ and we add noise using $\mathcal{N}(0, \sigma^2)$ with two different $\sigma = 0.1, 0.2$. For the ambient metric we use the cost based RBF approach by selecting 3 points and their 10 nearest neighbors in $\mathcal{X}$ with $y_k = 10$. We train two VAEs and we show in the latent space the resulting Riemannian metric with and without the stochasticity of the generator for the evaluation of the ambient metric $\mathbf{M}_{\mathcal{X}}(\cdot)$.

From the results in Fig. 12 we observe that by considering the true expected Riemannian metric, the captured structure does not differ significantly from the one we get using the proposed relaxation, especially, near the latent codes. Therefore, by taking into account the trade off, we argue that it is sufficient to use the expected generated manifold $\mathcal{M}_{\mathcal{Z}} = \mathbb{E}_{\varepsilon\sim p(\varepsilon)}[\mathcal{M}_{\varepsilon}]$ such that to evaluate the ambient Riemannian metric $\mathbf{M}_{\mathcal{X}}(\cdot)$ as it is shown in Eq. 21.

4.1 Overall approach and complexity

Our approach can be separated into two basic steps. In the first step, an ambient metric $\mathbf{M}_{\mathcal{X}}(\cdot)$ should be defined by the user, for example using one of the methodologies that we presented in this work. This allows to encode high level domain knowledge about the problem of interest in a geometric form. In some sense, we influence the shortest paths to "prefer" or to "avoid" regions of the ambient space. In the second step, we capture the geometry of the data manifold using the generative model, and in particular, the Jacobian of the generator $g(\cdot)$. The Jacobian of $g(\cdot)$ together with the ambient metric, induce a Riemannian metric in the latent space. The corresponding shortest paths in $\mathcal{Z}$ move optimally on the data manifold while respecting the geometry of the ambient space due to the $\mathbf{M}_{\mathcal{X}}(\cdot)$.

In principle, $g(\cdot)$ should be a stochastic mapping and we proposed a relaxation of the expected metric in the latent space (see Eq. 22). Also, we proposed one way that allows to properly capture the geometry in $\mathcal{Z}$ with deterministic models. This is based on an ambient metric that is small only near the given data, and the proposed architecture for $g(\cdot)$ that helps to extrapolate meaningfully. The later means that the $g(\cdot)$ becomes a linear map as we move further from the training latent codes in $\mathcal{Z}$. Of course, we can project the given data to a lower dimensional space and construct the ambient metric therein (see corresponding paragraph in Appendix 2). As regards the shortest paths, we can compute them solving the ODE system (Eq. 13) with numerical approximate solvers.

Obviously, the computational bottleneck of our approach is the evaluation of the Riemannian metric, and of course, the solution of the ODE system. For the metric we need to evaluate the $\mathbf{M}_{\mathcal{X}}(\cdot)$ as well as the Jacobian of $g(\cdot)$. The complexity of the ambient metric is based on its actual definition. Note that we can reduce the dimensionality of the ambient space metric by exploiting the linear projection step. In addition, we need to evaluate the $g(\cdot)$ such that to get the point in $\mathcal{X}$. Regarding, the Jacobian of $g(\cdot)$ this is a computationally expensive computation, since it is based on the architecture of $\mu(\cdot)$ which can be rather complicated. We can compute it using finite differences, automatic differentiation or code it analytically. Similarly, we can compute the Jacobian of the RBF network that is used for $\sigma(\cdot)$. Nevertheless, for evaluating

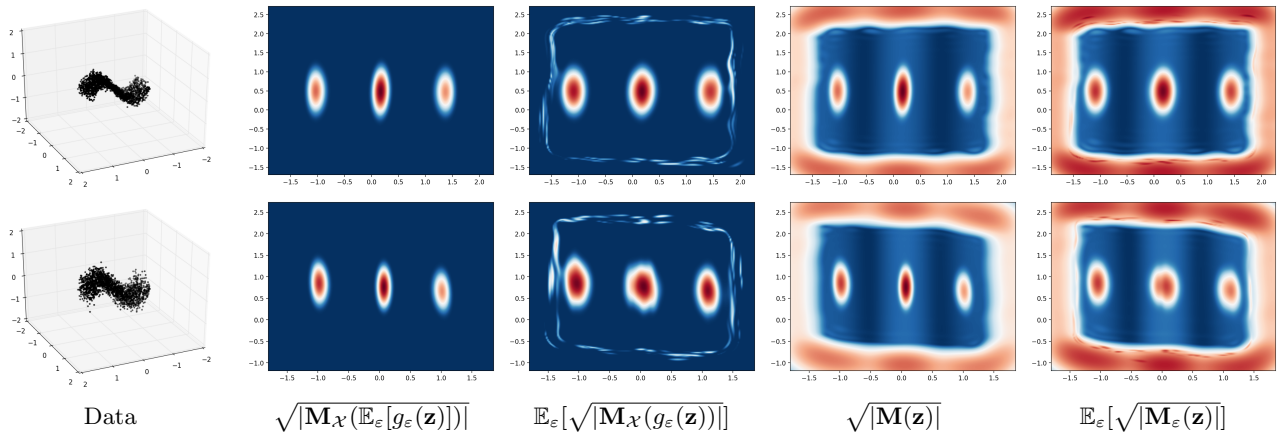


Figure 12: In the *top* row we added noise with $\sigma = 0.1$ and in the *bottom* row $\sigma = 0.2$. In the two last columns and per row, we see that the structure does not change significantly when we use the proposed relaxation.

the ODE system we additionally need the derivative of the Riemannian metric. This means that we need the derivative of the Jacobians, which increases more the computational complexity.

Having the metric and its derivative allows to evaluate the ODE system. Since analytic solution do not exist, we can utilize approximate numerical solvers in order to solve the system as a boundary value problem (BVP) and find the shortest path. However, the analysis of the computational complexity is difficult due to the iterative nature of the solvers.

As it is clear, our approach is computationally taxing for two reasons. The first is the Riemannian metric and its derivative. In this work we presented the direct way of combining the ambient metric with the pullback. One way to reduce the complexity is to approximate the enriched Riemannian metric with a surrogate metric that is computationally efficient. The second reason is the solution of the ODE system. Of course, improving the efficiency of the Riemannian metric directly improves the efficiency of the solvers. However, specialized approximate numerical BVP solvers can be developed such that to be faster and more robust (Arvanitidis et al., 2019; Hennig and Hauberg, 2014). Another way for finding the shortest path is by minimizing directly the energy using numerical differentiation and a parametric curve e.g. cubic-spline (Yang et al., 2018). The problem here is that we still rely implicitly on the Jacobian of $g(\cdot)$ that is commonly expensive, and also, we cannot solve the ODE system as an IVP to compute exponential maps. Regarding the heuristic solutions using the graph, this indeed, always finds a curve in the latent space. However, this curve in general does not satisfy the ODE system, so we cannot guarantee that this is the shortest path and also the resulting logarithmic map is arbitrary.

5 Experiments

In this section we provide further details and discussion regarding the conducted experiments.

5.1 Details for the Generative Adversarial Network demonstrations

Synthetic data. The synthetic data are generated as follows. First, we pick the centers of 6 Gaussian distributions $\mathcal{N}(\mu, 0.2^2 \cdot \mathbb{I}_2)$ uniformly on a circle with radius 3 and one in the center. Then, we generate 300 points from each Gaussian that can be seen as the actual latent representations, and thus, we construct the data $\mathbf{x} = [z_1, z_2, 0.3 \cdot (z_1^2 + z_2^2) + \varepsilon]$, where $\varepsilon \sim \mathcal{N}(0, 0.1^2)$. We used a Wassestein GAN with latent space $\mathcal{Z} = \mathbb{R}^2$ and ambient space $\mathcal{X} = \mathbb{R}^3$ and functions in Table 1.

We trained the model using Adam (Kingma and Ba, 2015) optimizer for 1000 epochs with stepsize $1e^{-2}$ and batch sizes of size 128, and also, we used ℓ_2 regularization for the weights with parameter $1e^{-5}$. The discriminator is trained for 5 more steps within each epoch and the weights are clamped into the interval $(-0.01, 0.01)$ to satisfy the Lipschitz constraint of the Wasserstein GAN. For the sampling of the latent codes we experimented both with standard Markon Chain Monte Carlo (MCMC), as well as rejection sampling (Bishop, 2006). For the mixture of LAND we used the default training procedure with 10 epochs and full covariance matrices per component. In order to construct the RBF ambient metric we used 20 components and the scaling factor of the bandwidth was set to $\kappa = 1$ as discussed in Appendix 3.2.

MNIST data. We used the digits 0,1,2, we scaled them in the interval $[-1, 1]$ and we added point-wise noise $\varepsilon \sim \mathcal{N}(0, 0.02^2)$, such that the data to not lie

Function	*Layer 1*	*Layer 2*	*Output*
$f(\mathbf{z})$	`tanh(2)`	`tanh(3)`	`linear(3)`
$d(\mathbf{x})$	`LeakyReLU(3) + Dropout(0.3)`	`LeakyReLU(3) + Dropout(0.3)`	`linear(1)`

Table 1: Functions for the GAN with synthetic data.

Function	*Layer 1*	*Layer 2*	*Output*
$f(\mathbf{z})$	`tanh(128)`	`tanh(256)`	`linear(784)`
$d(\mathbf{x})$	`tanh+LeakyReLU(128)+Drop(0.3)`	`LeakyReLU(128)+Drop(0.3)`	`linear(1)`

Table 2: Functions for the GAN with MNIST data.

exactly on $\mathcal{M}$. Thus, is easier to train the generator without utilizing the bounded $\texttt{tanh}(\cdot)$ in the output layer to clip the values. Otherwise, the meaningful extrapolation is not anymore useful, since the linear part will be also clipped. However, when pass the images into the critic $d(\mathbf{x})$ first we apply the $\texttt{tanh}(\cdot)$ function. The latent space is $\mathcal{Z} = \mathbb{R}^5$ and $\mathcal{X} = \mathbb{R}^{784}$ and the functions are defined in Table 2.

The discriminator is trained for 5 more steps within each epoch and the weights are clamped into the interval $(-0.01, 0.01)$ to satisfy the Lipschitz constraint of the Wasserstein GAN. The model is trained using Adam optimizer for 10000 epochs and batch size 64 with stepsize $1e^{-4}$ and ℓ_2 regularization of the weights with parameter $1e^{-7}$. For the sampling of the latent codes we experimented both with standard Markon Chain Monte Carlo, as well as rejection sampling. The ambient Riemannian metric is constructed with the RBF method discussed in Appendix 3.2 and we used 100 centers and $\kappa = 0.33$ which decreases the bandwidth of the RBF kernels. Moreover, we projected linearly the data to a lower dimensional space $\mathcal{X}' = \mathbb{R}^{10}$ using principal components analysis (PCA), where we constructed the metric $\mathbf{M}_{\mathcal{X}'}(\cdot)$ (see Appendix 2). In order to stabilize training and prevent mode collapse, we included a VAE loss with regularization parameter $1e^{-5}$, only when training the generator. Of course, this heuristic has mild influence to the generator.

We see that using the ambient metric $\mathbf{M}_{\mathcal{X}'}(\cdot)$ improves the sampling, and some additional results are shown in Fig. 14. The resulting samples due to the $\mathbf{M}_{\mathcal{X}'}(\cdot)$ lie closer to the support of the given data manifold, and also, we avoid samples in-between the disconnected components in $\mathcal{X}$. Moreover, we show some additional interpolations (see Fig. 15) where we again see that using the ambient metric improves the interpolations. In particular, the difference between our proposed approach and the standard shortest paths is that the ambient Riemannian metric pulls the paths towards the data manifold and avoids "shortcuts". Intuitively, shortcut means that the path moves optimally on the generated $\mathcal{M}_{\mathcal{Z}}$, but not necessarily always near the given data manifold. Note that $\mathcal{M}_{\mathcal{Z}}$ is a continuous smooth surface and some parts are not near the given data points/manifold, but without considering the $\mathbf{M}_{\mathcal{X}'}(\cdot)$ it might be "cheap" to move there which is a misleading behavior.

Pre-trained model. We used as generator a Progressive GAN (PGAN) (Karras et al., 2018) which utilizes a latent space $\mathcal{Z} = \mathbb{R}^d$ with $d = 512$ and has ambient space $\mathcal{X} = \mathbb{R}^D$ with $D = 256 \times 256 \times 3$, while the labeled training dataset is not directly provided. Note that in this generator it is not included the linear map to ensure meaningful extrapolation, and also, due to $\texttt{ReLU}(\cdot)$ activation the $\mathbf{M}(\cdot)$ is not sufficiently smooth. However, we tested how the additional consideration of an ambient metric can affect the shortest paths, and additionally, we use a heuristic that we describe below for computing approximate shortest paths where a smooth metric $\mathbf{M}(\cdot)$ is not necessary. Moreover, we upscaled the standard CelebA labeled dataset of size $128 \times 128 \times 3$ to D in order to be able to compute the linear projection matrix $\mathbf{P} \in \mathbb{R}^{d' \times D}$ from $\mathcal{X}$ to $\mathcal{X}'$ with $d' = 1000$ and the linear mean $\mathbf{c} \in \mathbb{R}^D$. See discussion in Appendix 2 regarding this linear projection step.

Obviously, the computation of the Jacobian matrix for this $g(\cdot)$ is prohibited due to the size of the latent space and the complexity of the model, even with finite differences. So we rely on the tricks explained below, in order to be able to compute relatively efficient shortest paths. First, we define a new latent space $\widetilde{\mathcal{Z}} = \mathbb{R}^{\tilde{d}}$ of dimensionality $\tilde{d} < d$ with $\tilde{d} = 10$ and we construct an ortho-normal random projection matrix $\widetilde{\mathbf{U}} \in \mathbb{R}^{d \times \tilde{d}}$. In such a way, we can compute shortest paths in $\widetilde{\mathcal{Z}}$ that correspond to shortest paths on a $\tilde{d}$-dimensional sub-space in $\mathcal{Z}$. Hence, in total we have

$$\widetilde{\mathcal{Z}} \xrightarrow{\widetilde{\mathbf{U}}\cdot} \mathcal{Z} \xrightarrow{g(\cdot)} \mathcal{X} \xrightarrow{\mathbf{P}\,(\cdot - \mathbf{c})} \mathcal{X}'. \tag{23}$$

Clearly, this tactic constraints the shortest paths to lie

on the linear subspace spanned by $\widetilde{\mathbf{U}}$ in $\mathcal{Z}$, and hence, they are not able to move freely in the whole $\mathcal{Z}$. However, this approximation allows us to compute shortest paths in reasonable time. Essentially, we induce the pull-back Riemannian metric in a lower dimensional latent space $\widetilde{\mathcal{Z}}$, while the $\widetilde{\mathbf{U}}$ matrix does not introduce further distortions as an orthonormal matrix.

The main reason for using the $\widetilde{\mathbf{U}}$ is that when $\tilde{d}$ is relatively small, we are able to compute the Jacobian matrix $\widetilde{\mathbf{J}}_g(\tilde{\mathbf{z}}) \in \mathbb{R}^{D\times\tilde{d}}$ using finite differences. In particular, in the latent space $\tilde{d}$ we approximate the j-th column of the Jacobian from $\widetilde{\mathcal{Z}} \to \mathcal{X}$ with finite differences as

$$\widetilde{\mathbf{J}}_g^j(\tilde{\mathbf{z}}) = \lim_{\lambda\to 0} \frac{g(\widetilde{\mathbf{U}}\cdot(\tilde{\mathbf{z}}+\lambda\mathbf{e}_j)) - g(\widetilde{\mathbf{U}}\cdot\tilde{\mathbf{z}})}{\lambda}, \tag{24}$$

where $\tilde{\mathbf{z}} \in \mathbb{R}^{\tilde{d}}$ and $\mathbf{e}_j = [0,\dots,1,\dots,0]$ a $\tilde{d}$-dimensional vector of zeros with 1 at the j-th location. Furthermore, we can exploit the forward pass to compute simultaneously all the columns of the Jacobian, by using a batch of inputs that we truncate using the identity matrix $\lambda\mathbb{I}_{\tilde{d}}$. In such a way, we can compute the Jacobian at a point with only one forward pass with batch size $\tilde{d}+1$. Nevertheless, even in an approximate ODE solver this is still very computationally expensive. So we implemented one heuristic to compute the shortest path based on the idea of ISOMAP (Tenenbaum et al., 2000).

We start by sampling 10000 points in $\tilde{d}$ uniformly inside a hypersphere of radius 4 and using k-means we find 100 prototypes. Thus, as we do not have access to latent codes, we introduce some artificial codes in $\widetilde{\mathcal{Z}}$. Then, using the prototypes we construct the K-nearest neighbor graph with $K=7$ by using the Euclidean distance to find the neighbors. But, for the weight of the edges we use the straight line distance measured under pull-back Riemannian metric that we can evaluate using the finite differences based Jacobian as

$$\begin{aligned}\text{length}[l(t)] &= \int_0^1 \sqrt{\langle \dot{l}(t_n), \mathbf{M}_{\text{fd}}(l(t_n))\dot{l}(t_n)\rangle}dt \\ &\approx \sum_{t_n=1}^{N} \sqrt{\langle \dot{l}(t_n), \mathbf{M}_{\text{fd}}(l(t_n))\dot{l}(t_n)\rangle}\Delta t_n,\end{aligned} \tag{25}$$

where $l(t)$ is the line between two latent points in $\widetilde{\mathcal{Z}}$. For the metric $\mathbf{M}_{\text{fd}}(\cdot)$ first we compute the Jacobian of the total map $\mathbf{P}(g(\widetilde{\mathbf{U}}\cdot l(t)) - \mathbf{c})$ with respect to $l(t)$, which can be achieved by using the finite differences for the Jacobian computation of the map $g(\widetilde{\mathbf{U}}\cdot l(t))$, and then, we use the $\mathbf{M}_{\mathcal{X}'}(\mathbf{P}(g(\widetilde{\mathbf{U}}\cdot l(t)) - \mathbf{c}))$. In particular the metric is equal to

$$\mathbf{M}_{\text{fd}}(\tilde{\mathbf{z}}) = \mathbf{A}^\intercal \mathbf{M}_{\mathcal{X}'}(\mathbf{P}(g(\widetilde{\mathbf{U}}\cdot\tilde{\mathbf{z}}) - \mathbf{c}))\mathbf{A} \tag{26}$$

$$\text{with} \quad \mathbf{A} = \left[\mathbf{P}\frac{\partial g(\widetilde{\mathbf{U}}\cdot\tilde{\mathbf{z}})}{\partial\tilde{\mathbf{z}}}\right]. \tag{27}$$

Essentially, the straight line in $\widetilde{\mathcal{Z}}$ measured under the Riemannian metric informs us how far on the manifold in $\mathcal{X}$ and under the metric $\mathbf{M}_{\mathcal{X}'}(\cdot)$ are the decoded latent points that seem to be close in the $\tilde{d}$-dimensions.

For two test points in $\widetilde{\mathcal{Z}}$ that we want to compute the shortest path, first we find their closest K-neighbors from the points on the graph using the Euclidean metric, and then, we assign the corresponding edge weights using the Riemannian distances. Finally, we chose two auxiliary points, one per k-NN set with the smallest Riemannian distance. Thus, we can find the discrete shortest path using Dijkstra's algorithm on the graph using the auxiliary nodes as the boundary points. Note that the path prefers edges with low weight i.e., the edge corresponds to a curve on $\mathcal{M}$ with small length. Ultimately, for the continuous path we use the cubic spline interpolation through the points of the discrete path on the graph replacing the two auxiliary points with the test points. Obviously, this heuristic method approximates the true shortest path, but the result does not satisfy the ODE system. This approach is inspired by ISOMAP and a very similar heuristic approach has been proposed in Chen et al. (2019).

The task that we want our ambient Riemannian metric to model, is to avoid regions with blond people when interpolating between two latent codes. As we described above we linearly project in $\mathcal{X}'$ the implicitly given data manifold $\mathcal{M} \subset \mathcal{X} = \mathbb{R}^D$, by using the standard labeled CelebA dataset. In $\mathcal{X}'$, we construct the $\mathbf{M}_{\mathcal{X}'}(\cdot)$ which is based on a simple RBF cost based metric (see Appendix 3.3) with $y_k = 1e^9$ and $\sigma = 5$. Therefore, we have to define the centers $\mathbf{c}_k \in \mathcal{X}'$. In order to do that, first we train on the labeled CelebA dataset of size $128\times128\times3$ a simple convolutional neural network classifier $c(\mathbf{x})$ (see Table 3). Once the classifier is trained, we decode the nodes of the graph and samples from the prior, which we classify after resizing from $\mathbb{R}^{256\times256\times3}$ to $\mathbb{R}^{128\times128\times3}$. With these steps, we are able to define the centers of the metric in $\mathcal{X}'$, by using the points that are classified as blond. Note that this is a very simple to implement metric, but rather informative, since the shortest path is penalized heavily when moves close to the high cost regions in $\mathcal{X}'$. Essentially, the (discrete) shortest path avoids the nodes which after decoding fall near the high cost regions in $\mathcal{X}'$. We show some further interpolation results in Fig. 16 using different projection matrices $\widetilde{\mathbf{U}}$, which means that we explore different subspaces in $\mathcal{Z}$, and consequently, on $\mathcal{M}_{\mathcal{Z}}$.

Function	*Layer 1,2,3*	*Output*
$c(\mathbf{x})$	$3 \times$ [`conv(16,5,1)` + `MaxPool(2)` + `ReLU`]	`Sigmoid(Linear(265, 1))`

Table 3: The classifier that we used in the Progressive GAN experiment.

5.2 Details for the Variational Auto-Encoder experiments

We used the MNIST digits 0,1,2,3 scaled in the interval $[-1, 1]$ and then we added point-wise noise $\varepsilon \sim \mathcal{N}(0, 0.02^2)$. As we explained before, we add the noise such that the data to not lie exactly on $\mathcal{M}$, so that we can train the generator without utilizing the bounded $\texttt{tanh}(\cdot)$ in the output layer to clip the values. Note that in the stochastic generator case the meaningful extrapolation is not necessary, even if we use it in our experiments, since the uncertainty quantification helps to properly capture the data manifold structure. The ambient space is $\mathcal{X} = \mathbb{R}^{784}$ and the latent space $\mathcal{Z} = \mathbb{R}^5$. We used the functions in Table 4, where the $\beta(\mathbf{z}) + \zeta = \frac{1}{\sigma^2(\mathbf{z})}$ with $\zeta = 1e^{-6}$ and $\beta(\mathbf{z})$ is an RBF with 100 centers and only positive weights. We trained the model using Adam optimizer for 1000 epochs and batch size 64 with stepsize $1e^{-4}$ and also ℓ_2 regularization of the weight with parameter $\lambda = 1e^{-5}$.

For the interpolation experiment, the LDA metric is constructed by considering the digtis 0,1,3 in the same class, while in the kernel density estimation experiment every class is separated. We used 2000 randomly chosen training points as the base points $\mathbf{x}_s$, the $\varepsilon = 1e^{-3}$, the number of nearest neighbors is $K = 50$ and we used a fixed number of iterations 20. See Appendix 3.1 for details.

For the cost function based ambient metrics we use the RBF cost discussed in Appendix 3.3. We start by picking 3 latent codes in $\mathcal{Z}$ and we decode them. Then, we find the closest 100 neighbors per decoded point in $\mathcal{X}'$ and we construct the metric with parameters $y_k = 100$ and $\sigma = 0.2$. Note that we find the neighbors using the Euclidean distance. So in total we have 300 RBF basis functions in $\mathcal{X}'$. We used the same approach both in the interpolations and the KDE experiment.

For the linear combination of the ambient metrics we used the weights 1 for the LDA, 0.001 for the local diagonal inverse covariance and 0.1 for the cost metric. We used the same coefficients both in the interpolations and the KDE experiment. Also, the reason for so different coefficients is the scaling of each individual metric. Of course, choosing carefully the parameters of each ambient metric could regularize the scaling differences. However, a principled method to estimate the mixture coefficients is a future problem.

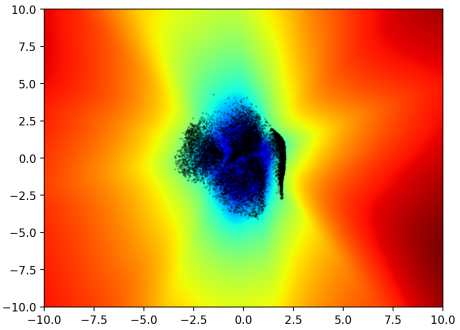
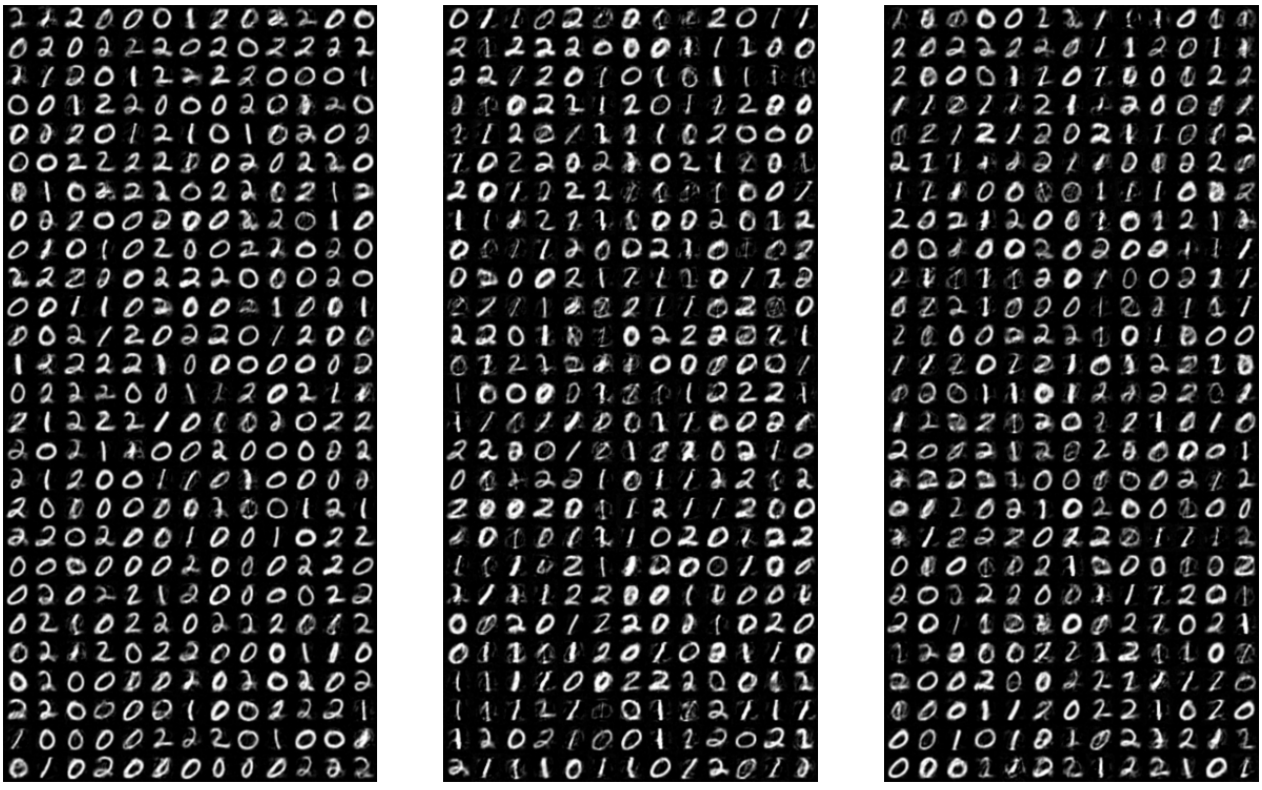


Figure 13: Meaningful extrapolation. The distance $\|\mathbf{b} - \mu(\mathbf{z})\|_2$ is computed in $\mathcal{X}$ and presented in $\mathcal{Z}$, where $\mathbf{b}$ is the center of the data.

As an additional experiment we examine if the proposed meaningful extrapolation technique is actually working. Therefore, using a set of points $\mathbf{z}_s$ on a uniform grid in the latent space, we generate the points in $\mathcal{X}$ on the expected manifold $\mathcal{M}_{\mathcal{Z}}$ as $\mathbf{x}_s = \mu(\mathbf{z}_s) = f(\mathbf{z}_s) + \mathbf{U} \cdot \text{diag}([\sqrt{\lambda_1}, \sqrt{\lambda_2}]) \cdot \mathbf{z}_s + \mathbf{b}$. Here, the $f : \mathcal{Z} \to \mathcal{X}$ is a DNN and the linear part is defined as explained in the main paper. In Fig. 13 we show for each $\mathbf{z}_s$ the Euclidean distance measured in $\mathcal{X}$ between the center of the training data and the corresponding point $\mathbf{x}_s$. Indeed, we see that as we move further from the prior $p(\mathbf{z})$ support, the Euclidean distance between the points on the generated surface and the center of the data increases. However, we observe that the distance on the x_1-axis increases faster than the x_2-axis. The reason is that the corresponding eigenvalue of the linear map is higher, so the generated $\mathcal{M}_{\mathcal{Z}}$ extrapolates linearly faster along this latent dimension. Note that in this example we used the $\texttt{softplus}(\cdot)$ activation function, for which the extrapolation behavior is more difficult to analyze than the $\texttt{tanh}(\cdot)$. Even so, we get a meaningful extrapolation due to the linear part of the function $\mu(\cdot)$.

Function	*Layer 1*	*Layer 2*	*Output*
decoder: $f(\mathbf{z})$	`softplus(128)`	`softplus(256)`	`linear(784)`
decoder: $\beta(\mathbf{z})$	`RBF(100)`		`linear(784)`
encoder: $\mu_\phi(\mathbf{x})$	`softplus(256)`	`softplus(128)`	$\texttt{linear}_{>0}\texttt{(5)}$
encoder: $\sigma^2_\phi(\mathbf{x})$	`softplus(256)`	`softplus(128)`	`softplus(linear(5))`

Table 4: The functions used in the VAE experiments.



(a) From $q(\mathbf{z})$ with $\mathbf{M}_{\mathcal{X}}(\cdot)$. (b) From $q(\mathbf{z})$ without $\mathbf{M}_{\mathcal{X}}(\cdot)$. (c) From prior $p(\mathbf{z})$.

Figure 14: Additional samples for the MNIST data, GAN experiment with 5-dimensional latent space. Note that our proposed method does not generate a lot of ghostly samples, which fall on parts of the ambient space with no given data nearby.

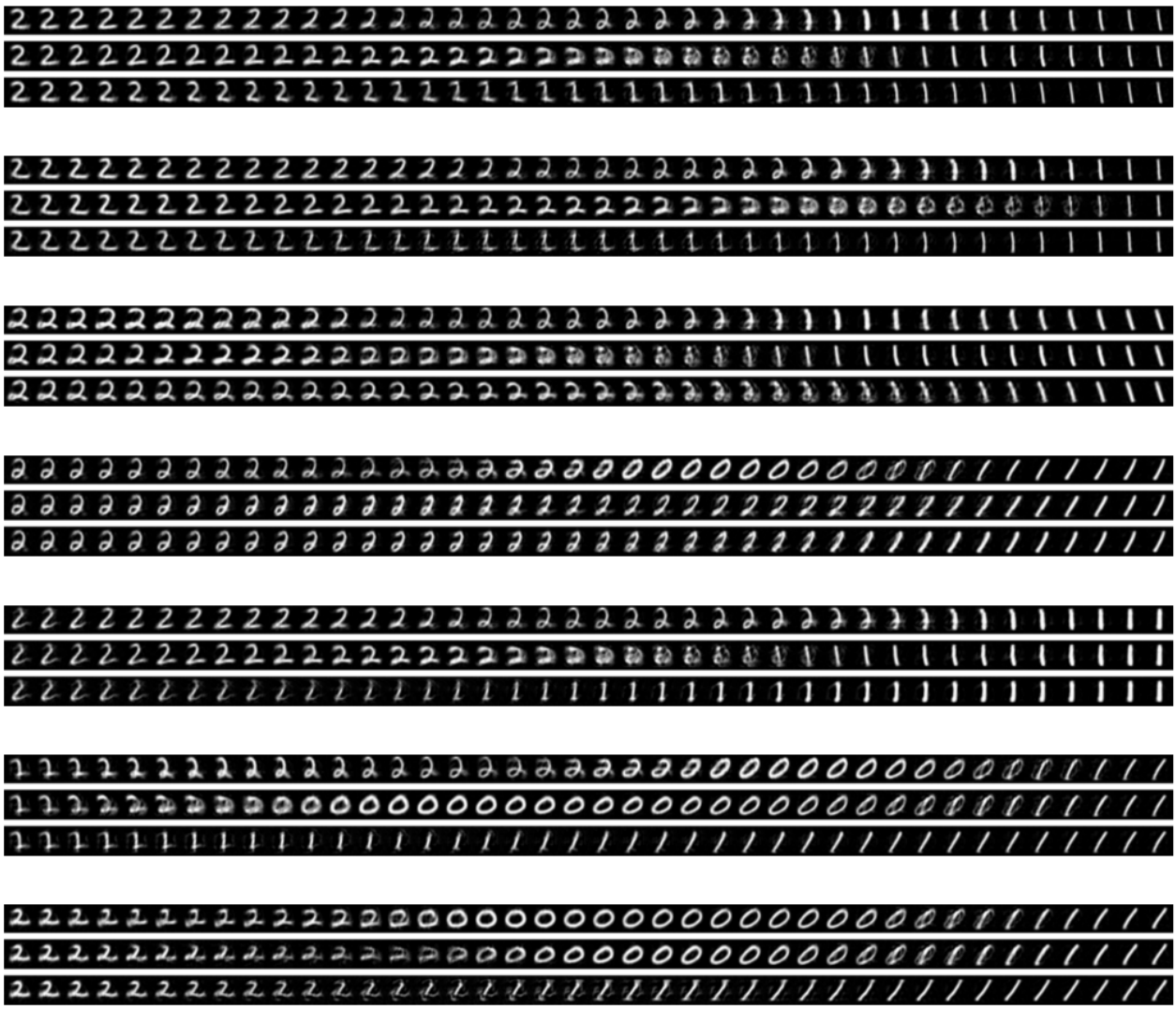


Figure 15: Additional interpolation results. From *top* to *bottom*: our interpolation, interpolation without using the $\mathbf{M}_{\mathcal{X}}(\cdot)$, linear interpolation. Note that our interpolation (top rows) avoids the "shortcuts" of the simple shortest path interpolant (middle rows), while the linear path is arbitrary.

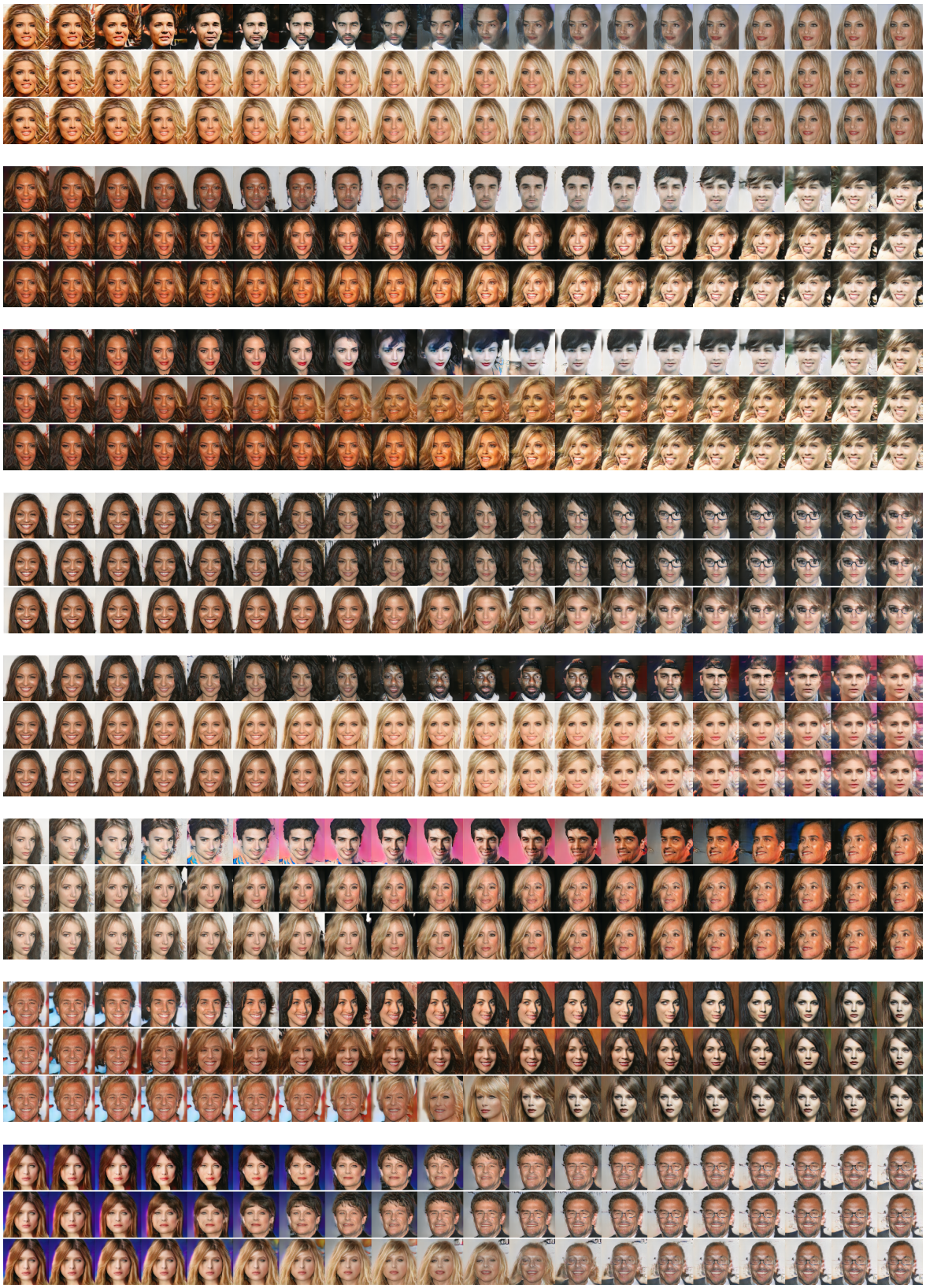
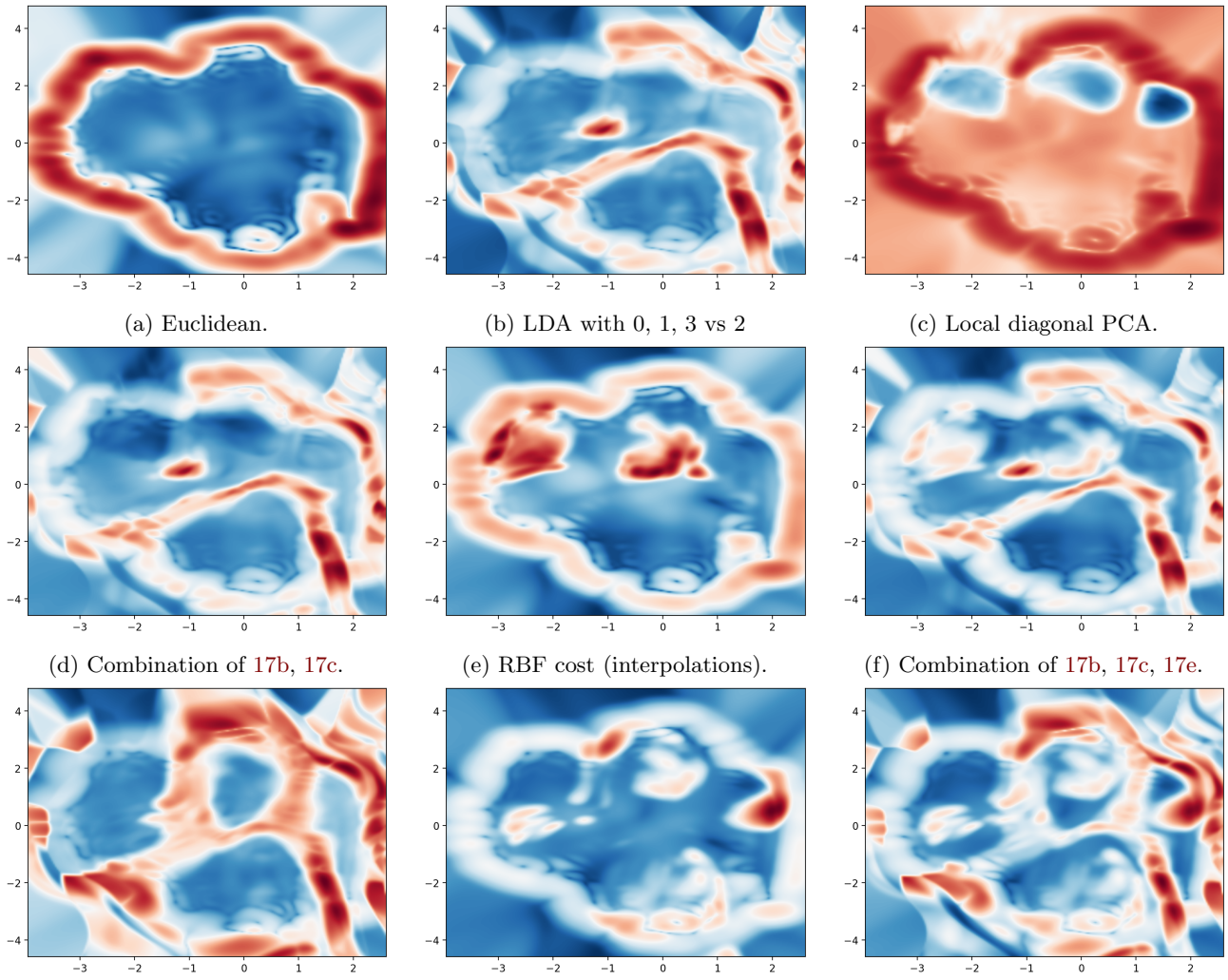


Figure 16: Additional interpolation results for the PGAN. From *top* to *bottom*: our interpolation, interpolation without using the $\mathbf{M}_{\mathcal{X}'}(\cdot)$, linear interpolation. Note that our interpolation (top rows) provides a smooth transition between the images, while it avoids the high cost regions (people with blond hair). The shortest path without the ambient metric still provides smooth changes, however, it often crosses high cost regions. The relatively smooth behavior of the straight line is due to the nature of the generator and not due to the actual interpolant.



(a) Euclidean. (b) LDA with 0, 1, 3 vs 2 (c) Local diagonal PCA.

(d) Combination of 17b, 17c. (e) RBF cost (interpolations). (f) Combination of 17b, 17c, 17e.

(g) LDA separate 0,1,2,3. (h) RBF cost (KDE experiment). (i) Combination of 17g, 17h.

Figure 17: The Riemannian measure for the VAE experiments and several ambient metrics $\mathbf{M}_{\mathcal{X}}(\cdot)$. In each caption we mention briefly the form and the details of the ambient metric.

References

Arvanitidis, G., Hansen, L. K., and Hauberg, S. (2018). Latent space oddity: on the curvature of deep generative models. In *International Conference on Learning Representations (ICLR)*.

Arvanitidis, G., Hauberg, S., Hennig, P., and Schober, M. (2019). Fast and robust shortest paths on manifolds learned from data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*.

Chen, N., Ferroni, F., Klushyn, A., Paraschos, A., Bayer, J., and van der Smagt, P. (2019). Fast approximate geodesics for deep generative models. *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*.

Eklund, D. and Hauberg, S. (2019). Expected path length on random manifolds. In *arXiv preprint*.

Hastie, T. and Tibshirani, R. (1994). Discriminant adaptive nearest neighbor classification.

Hauberg, S. (2018). Only bayes should learn a manifold.

Hennig, P. and Hauberg, S. (2014). Probabilistic Solutions to Differential Equations and their Application to Riemannian Statistics. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018). Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*.

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations, ICLR*.

Suárez, J. L., García, S., and Herrera, F. (2018). A tutorial on distance metric learning: Mathematical foundations, algorithms and experiments.

Tenenbaum, J. B., de Silva, V., and Langford, J. C. (2000). A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*.

Yang, T., Arvanitidis, G., Fu, D., Li, X., and Hauberg, S. (2018). Geodesic clustering in deep generative models. In *arXiv preprint*.