Capturing topological properties of the rotation group using variational auto-encoders

Bon, Daan L.J.

Award date:
2020

Eindhoven University of Technology

Capturing topological properties of the rotation group using variational auto-encoders

Author: D.L.J. Bon

Supervisor: J.W. Portegies

Juli 2020

Abstract

In this paper, we investigate the capabilities of the Diffusion Variational Auto-Encoder (ΔVAE) with a $SO(3)$ latent space to learn meaningful latent representations of data with a $SO(3)$ latent structure. To investigate the behaviour when training a ΔVAE on such a data set, we make use of the concept of the degree of a mapping. We calculate the degree explicitly for certain functions, and present a method to compute it for a more general class. Lastly, we run several experiments on a synthetic data set, and adapt the ΔVAE to be able to learn better representations of the latent variables.

Contents

1 Introduction

Unsupervised learning is an important aspect of machine learning. A majority of the data available is unlabeled and not suitable for supervised learning techniques, as labelling data usually requires human interaction, making it a time-consuming process. One especially interesting part of unsupervised learning is the extraction of latent variables. Latent variables are variables inferred from the observable data. They are of interest because they are intimately tied to the data, and they might reveal a structure which was not immediately obvious by looking at the data alone. Next to that, the fact that latent variables are often low-dimensional, compared to the dimensionality of the data, also helps in computations.

There are many techniques which try to extract meaningful latent variables, think for example of principal component analysis. One relatively new technique uses the Variational Auto-Encoder (VAE) [12] [17].

Variational auto-encoders consist of a latent space Z, a probability measure $\mathbb{P}_Z$ on Z, a family of encoder conditional distributions $q_\phi(\mathbf{z}|x)$ over Z, and a family of decoder distributions $\mathbb{P}_\theta(x|\mathbf{z})$ over X. We then aim to find good parameters for these encoder/decoder distributions, usually by using neural networks and gradient descent, to minimize the negated evidence lower bound (ELBO):

$$-\mathcal{L}(x) = -\mathop{\mathbb{E}}_{z\sim q_\phi}[\log(\mathbb{P}_\theta(x))] + KL(q_\phi||\mathbb{P}_Z).$$

The original VAE has been adapted by several authors to increase its capability of learning a good latent representation [9], [2], [11]. However, a drawback is that standard variational auto-encoders have a Euclidean latent space. They are thus incapable of capturing certain latent structures data might have. More concretely, the latent variables might have a non-trivial topology, which means that there is no homeomorphism between it and Euclidean space. As an example of a data set with a non-Euclidean latent variable consider images of an object rotated around a fixed axis. The latent variable is the angle of rotation, which has a circular structure. So its latent space is homeomorphic to S^1, which is not homeomorphic to any $\mathbb{R}^d$. This phenomenon has been called *manifold-mismatch* [3].

Several solutions to this problem have been proposed. In [4] Falorsi et al. construct VAEs which have Lie groups as latent space. In particular, they show how the reparametrization trick can be adapted to $SO(3)$. In [3] Davidson et al. construct VAEs which have a hyperspherical latent space. In [16] Perez et al. propose the Diffusion VAE, or ΔVAE , which can have an arbitrary Riemannian manifold as latent space. However, as highlighted by an example in the paper, for more complex synthetic data sets generated with a particular latent structure, the ΔVAE is not capable of fully capturing this structure.

This report investigates the ΔVAE , with the $SO(3)$ as latent space. In particular, for synthetic data sets with a latent structure of the $SO(3)$ the ΔVAE sometimes has trouble detecting this structure. For more complex, real life data sets it is unable to detect structure at all. By looking into why things go wrong in the synthetic data setting, we hope to get a better understanding of the workings of the ΔVAE . In order to investigate the behaviour of the network, the concept of the degree of a mapping will be used. This will first be introduced, after which we attempt to compute the degree of the decoder part of the VAE. Lastly we will train the ΔVAE on a data set consisting of $SO(3)$ matrices, and a more "real-life" data set in order to see if we can increase its capabilities of capturing the topological properties of the data.

2 Preliminaries

Before discussing VAEs and the ΔVAE in more detail, we will first summarize some important notions needed for later sections of the report. In particular, the degree of a mapping and the rotation group $SO(3)$ are discussed in detail.

2.1 Mapping degree

We start by discussing the notion of the degree for functions between smooth manifolds. This will be an important tool we use to study the ΔVAE later on. We will follow the explanations given by Milnor in [15]. However, we choose to give a more concise summary of the earlier part of these notes which have to do with introducing smooth manifolds and smooth functions between them. Where necessary only the relevant definition will be given. For another introduction to smooth manifolds, see [13].

2.1.1 Smooth mappings and smooth manifolds

For reference we add the relevant definitions having to do with smooth manifolds and mappings.

We call a mapping from an open set $U \subset \mathbb{R}^n$ to an open set $V \subset \mathbb{R}^m$ smooth if all of its partial derivatives exist and are continuous (a C^∞ function). We call a function f between U and V a *homeomorphism* if it is continuous and its inverse exists and is continuous as well. If f and f^{-1} are moreover smooth, we call f a *diffeomorphism*.

Definition 1 ([15], page 1)**.** *A map $f : U \subset \mathbb{R}^n \to V \subset \mathbb{R}^m$ is called smooth if for each $x \in U$ there exist an open set $X \subset \mathbb{R}^n$, containing x, and a function $F : X \to \mathbb{R}^m$ such that f coincides with F on $U \cap X$ and F is smooth.*

Definition 2 ([15], page 1)**.** *A set $M \subset \mathbb{R}^n$ is called a submanifold of $\mathbb{R}^n$, of dimension d, if each $x \in M$ has a neighborhood $W \cap M$ that is diffeomorphic to an open subset U of $\mathbb{R}^d$.*

Definition 3 ([15], page 4)**.** *Let M be a submanifold of dimension d and $x \in M$. Let $g : U \to M$ parametrize some neighborhood $g(U)$ of x, with $u \in U$ such that $g(u) = x$. The tangent space of M at x, denoted TM_x, is defined by $TM_x = dg_u(\mathbb{R}^d)$. Here dg_u denotes the derivative of g in the point u.*

Notice that the tangent space also has dimension d. Now imagine we have two submanifolds, $M \subset \mathbb{R}^m$ and $N \subset \mathbb{R}^n$, and a smooth map $f : M \to N$. Let $x \in M$ and $y \in N$ such that $f(x) = y$. Then the derivative $df_x : TM_x \to TN_y$ is defined as follows.

Definition 4 ([15], page 6)**.** *Because f is smooth there exists a open set W containing x and a function $F : W \to \mathbb{R}^n$ that coincides with f on the neighborhood $W \cap M$ around x. The derivative $df_x(h)$ is defined to equal $dF_x(h)$ for all $h \in TM_x$.*

Let C be the set of all $x \in M$ such that df_x has rank less than dim (N). Then C is called the set of *critical points*, the complement $M - C$ the set of *regular points*, $f(C)$ the set of *critical values* and the complement $N - f(C)$ the set of *regular values* of f.

Definition 5 ([15], page 20)**.** *Two mappings $f, g : X \to Y$ are called smoothly homotopic (denoted $f \sim g$) if there exists a smooth map $F : X \times [0, 1] \to Y$ with $F(x, 0) = f(x)$ and $F(x, 1) = g(x)$ for all $x \in X$. F is called a smooth homotopy.*

Notice that the relation of smooth homotopy is an equivalence relation on functions.

2.1.2 Degree of a smooth mapping

Now that the basic concepts are defined, we turn to the degree of a mapping. In order to define it, we first define what an orientation of a manifold is. To this end, we first look at orientations on vector spaces.

Definition 6 ([15], page 26)**.** *An orientation for a finite dimensional real vector space is an equivalence class of ordered bases. A ordered basis* $(b_1, \cdots, b_n)$ *determines the same orientation as the basis* $(c_1, \cdots, c_n)$ *if* $c_i = \sum a_{i,j} b_j$ *where* $A = a_{i,j}$ *has* $\det(A) > 0$*. It determines the opposite orientation if* $\det(A) < 0$*.*

The vector space $\mathbb{R}^n$ has a standard orientation corresponding to the standard basis given to this vector space: $(e_1, \cdots, e_n)$. Notice that a vector space has exactly two equivalence classes for orientation.

An oriented submanifold of $\mathbb{R}^n$ consists of a submanifold M together with a choice of orientation for each tangent space TM_x. These need to satisfy the following: for each $x \in M$ there should exist a neighborhood $U \subset M$ and a diffeomorphism h mapping U in a open subset of $\mathbb{R}^d$ which *preserves orientation*, meaning that for each $u \in U$ the isomorphism dh_u carries the specified orientation for the tangent space TM_u into the standard orientation for $\mathbb{R}^d$.

An oriented manifold is then a manifold together with a choice of orientation for each tangent space, fitting together as described above. We say a smooth manifold M is *orientable* if we can find such orientations for the tangent spaces of M. There are some manifolds which are not orientable, the most famous being the Möbius strip. If M is connected and orientable, it has precisely two orientations.

We are now ready to define the degree of a mapping. Let M and N be oriented n-dimensional manifolds and $f : M \to N$ a smooth map. Then we have the following definition for the degree of f.

Definition 7 ([15], page 27)**.** *Let* $x \in M$ *be a regular point of* f*. Define the sign of* Df_x *to be* $+1$ *or* -1 *according to if* Df_x *preserves of reverses orientation. Then for any regular value* $y \in N$*:*

$$deg\ (f; y) = \sum_{x \in f^{-1}(y)} sign\ Df_x.$$

We have the following two important theorems about the degree:

Theorem 2.1 ([15], Theorem A)**.** *The value deg* $(f; y)$ *does not depend on the choice of regular value* y*.*

Theorem 2.2 ([15], Theorem B)**.** *If* $f \sim g$*, then* $\deg(f) = \deg(g)$*.*

This justifies the notation deg (f) for the degree of a function f, without a particular choice for y.

2.1.3 Other views on the degree

The degree is something that can be defined in a number of different ways. One of these ways is in the context of *homology*. For a general reference about homology see [8], where Section 2.2 is about the degree. In homology, the functions considered are continuous, not necessarily differentiable or smooth. The definition for the degree is as follows.

Definition 8. *Let* M, N *be orientable manifolds of the same dimension,* d*. Let* $f : M \to N$ *be continuous. Then the degree of* f*, deg* (f) *is the unqiue integer* $k \in \mathbb{Z}$ *such that* $f_*(x) = kx$*, where*

$$f_* : H_d(M) \to H_d(N)$$

is the map induced by f *on the top homology groups.*

At first glance this does not resemble Definition 7. However, there are ways to compute the degree in homology which look more like the definition presented here, for example Proposition 2.30 in [8].

2.2 The $SO(3)$

As we will consider the case where we train the ΔVAE on a $SO(3)$ latent space, we introduce this space in some more detail. In particular, we look at its tangent space, and several properties of the projection on the closely related $O(3)$.

Definition 9. *We denote the space of 3×3 matrices, $\mathbb{R}^{3\times3}$, by $\mathcal{M}_3$.*

Definition 10. *$GL(n)$ denotes the general linear group of $n \times n$ matrices with coefficients in $\mathbb{R}$. This is the set of invertible matrices, or matrices with non zero determinant. We let $GL^+(n)$ denote the connected component of $GL(n)$, of matrices with positive determinant. Similarly let $GL^-(n)$ denote the set of matrices with negative determinant.*

Any rotation and reflection of three dimensional euclidean space can be represented by a 3×3 orthogonal matrix (a matrix O such that $OO^T = I$). The set of all orthogonal 3×3 matrices together with matrix multiplication form a well-know group, the *orthogonal group* or $O(3)$. This group has two connected components, namely the matrices with determinant -1 and with determinant 1. The subgroup corresponding to the set of orthogonal matrices with determinant 1 is called the *special orthogonal group* or *rotation group*, denoted as $SO(3)$. It represents all linear transformations of $\mathbb{R}^3$ that preserve orientation and length (also called true rotations). Besides being a group, it also has the structure of a smooth manifold. This makes it a so-called *Lie group*. For a general reference on Lie groups see [18] or [7], where [18] is a more introductory source.

2.2.1 The tangent spaces of the $SO(3)$

The first thing we will do is look at the tangent spaces of the $SO(3)$. We first look at the tangent space of the identity, as it will turn out that for Lie groups that is all that is necessary. The tangent space at the identity for Lie groups is also called the *Lie algebra*. See [18] Section 5, in particular Section 5.2.

Consider some curve on $SO(3)$, denoted by $R : [-1, 1] \to SO(3)$, where $R(0) = I$. Then we know that for each $t \in [-1, 1]$, $R(t)R(t)^T = I$. We now differentiate this curve, and see that its derivative needs to satisfy:

$$R(t)\frac{dR(t)}{dt}^T + \frac{dR(t)}{dt}R(t)^T = 0.$$

Substituting $t = 0$ into this equation, we find that the tangent vectors, $X = \frac{d}{dt}R(0)$, at the identity need to satisfy:

$$X = -X^T,$$

which are matrices known as anti-symmetric matrices. They form a three-dimensional vector space, and thus form the complete tangent space of the $SO(3)$ at the identity, $TSO(3)_I$. We choose the following basis for this vector space:

$$L_x = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}, \quad L_y = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix}, \quad L_z = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

We choose to orient the tangent space at the identity according to the following ordered basis:

$$\big(L_x,\ L_y,\ L_z\big).$$

We would also like to know what the other tangent spaces look like. Let $O \in SO(3)$. Then, using a similar method as above, the tangent vectors $X \in TSO(3)_O$ need to satisfy:

$$OX^T + XO^T = 0.$$

There is an isomorphism between the tangent space at the identity and the tangent space at O, which is the linearization of $L_O : SO(3) \to SO(3)$, left translation by O. This linearization looks like:

$$DL_O : TSO(3)_I \to TSO(3)_O$$
$$DL_O : X \mapsto OX.$$

We can see that matrices of this form, OX, for $X \in TSO(3)_I$ indeed satisfy the requirement found above:

$$\begin{aligned} O(OX)^T + (OX)O^T &= OX^TO^T + OXO^T \\ &= -OXO^T + OXO^T \\ &= 0 \end{aligned}$$

And so we can transform our basis for the tangent space at I to any other tangent space on $SO(3)$ by left translation of the basis. Moreover, we can choose a consistent orientation of $SO(3)$ in this way. Let $O \in SO(3)$, then the tangent space $TSO(3)_O$ has orientation given by:

$$\big(OL_x,\, OL_y,\, OL_z\big)$$

2.2.2 Projecting on the $O(3)$

For the ΔVAE we need to project back to the closest point on the manifold (see Section 3.2.1). In other words, given some arbitrary matrix M, we wish to find the closest special orthogonal matrix O. We first consider projecting on $O(3)$.

So we are interested in the function $P : GL(3, \mathbb{R}) \to O(3)$ assigning to each matrix its closest orthogonal matrix. This is known as the Procrustes problem, see [6] Section 6.4.1. P can be given in terms of the polar decomposition. Every $M \in \mathcal{M}_3$ can be written as the product of a orthogonal matrix O and a symmetric positive definite matrix S, $M = OS$. Then we have for $M \in GL(3)$ that:

$$P(M) = O \tag{1}$$

The reason P is only defined on $GL(3)$ and not on the whole of $\mathcal{M}_3$ is because the polar decomposition is not unique in the case where M has determinant zero.

The way O is calculated in the polar decomposition lets us write the function in the following equivalent forms as well:

$$P(M) = M(M^TM)^{-\frac{1}{2}}$$

or given in terms of the Singular Value Decomposition (SVD), where $M = U\Sigma V^T$:

$$P(M) = UV^T$$

We will use these different forms interchangeably. This projection has some properties we will use. Let O be an orthogonal matrix, and $M \in \mathcal{M}_3$. Then we have:

Lemma 1. $P(MO) = P(M)O$

Proof.

$$\begin{aligned} P(MO) &= MO((MO)^T MO)^{-\frac{1}{2}} \\ &= MO(O^T M^T MO)^{-\frac{1}{2}} \\ &= MOO^T (M^T M)^{-\frac{1}{2}} O \\ &= M(M^T M)^{-\frac{1}{2}} O \\ &= P(M)O \end{aligned}$$

We can take the O outside of the inverse square root by noticing that because $\det(M) > 0$ $\det(O^T M^T MO) > 0$, and $O^T M^T MO$ is symmetric positive definite, so its inverse square root is unique. It is the unique matrix K such that $O^T M^T MO(K^2) = I$. Then checking for $K = O^T (M^T M)^{-\frac{1}{2}} O$ we find that indeed $O^T M^T MO(K^2) = I$. □

Lemma 2. $P(OM) = OP(M)$

Proof.

$$\begin{aligned} P(OM) &= OM((OM)^T OM)^{-\frac{1}{2}} \\ &= OM(M^T O^T OM)^{-\frac{1}{2}} \\ &= OM(M^T M)^{-\frac{1}{2}} \\ &= OP(M) \end{aligned}$$

□

2.2.3 Derivative of the projection

We are interested in the derivative of this projection, which is needed for the so called reparametrization trick, see Section 3.2.1. For a general reference about matrix functions see [10], where Chapter 3.2 is about the Fréchet derivative. This reference states a number of properties we will use. First of all, the definition as reminder.

Definition 11. *We say a function $F : \mathcal{M}_3 \to \mathcal{M}_3$ is (total, Fréchet) differentiable in some point $M \in \mathcal{M}_3$ if there exists a linear map:*

$$(DF)_M : \mathcal{M}_3 \to \mathcal{M}_3$$

such that for all $K \in \mathcal{M}_3$:

$$F(M + K) = F(M) + (DF)_M(K) + o(||K||),$$

The map $(DF)_M(K)$ is then called the (total, Fréchet) derivative in M in the direction of K.

Let $(D[f(S)])_M(N)$ denote the derivative of the function $f : \mathcal{M}_3 \to \mathcal{M}_3$ in M in the direction of N, as defined above. So for example $(D[S + S^T])_M(N)$ denotes the derivative of the function $f : \mathcal{M}_3 \to \mathcal{M}_3$, given by $f(S) = S + S^T$.

In order to calculate the derivative of the projection function, equation (1), we will first calculate the derivative of several other functions.

Proposition 1. *The derivative of the function $f : \mathcal{M}_3 \to \mathcal{M}_3$, given by $f(M) = M^T M$ is:*

$$(Df)_M(N) = N^T M + M^T N$$

Proof. We use the product rule. We have that:

$$(Df)_M(N) = (D[S^T])_M(N)M + M^T(D[S])_M(N)$$

Both the identity function and the transpose function are linear, and so they are equal to their own derivative. So we find that:

$$(Df)_M(N) = N^T M + M^T N$$

□

Proposition 2. *The derivative of the function $g : GL(3) \to GL(3)$, given by $g(M) = M^{-1}$ is:*

$$(Dg)_M(N) = -M^{-1}NM^{-1}$$

Proof. Let id be the identity function on $GL(3)$. Then we have that $g(M)id(M) = I$ for every $M \in GL(3)$. We now differentiate both sides to get:

$$\begin{aligned}(Dg)_M(N)id(M) + g(M)(D(id))_M(N) &= 0\\ (Dg)_M(N)M + M^{-1}N &= 0\\ (Dg)_M(N) &= -M^{-1}NM^{-1}\end{aligned}$$

□

Proposition 3. *Let $h : GL(3) \to GL(3)$ be given by $h(M) = M^{\frac{1}{2}}$. Then $(Dh)_M(N)$ is the solution to equation:*

$$M^{-\frac{1}{2}}(Dh)_M(N) + (Dh)_M(N)M^{-\frac{1}{2}} = N$$

Proof. Let $h^{-1} : GL(3) \to GL(3)$ be given by $h^{-1}(M) = M^2$. Then $h^{-1} \circ h = id$. We differentiate this equation to find:

$$\begin{aligned}(Dh^{-1})_{M^{\frac{1}{2}}}((Dh)_M(N)) &= N\\ M^{\frac{1}{2}}(Dh)_M(N) + (Dh)_M(N)M^{\frac{1}{2}} &= N\end{aligned}$$

□

Equations of the form $AX + XB = C$ are known as *Sylvester equations*. In our particular case, we are only interested in evaluating the derivative in matrices of the form $M^T M$, where $M \in GL(3)$. These are positive symmetric matrices. If $M = U\Sigma V^T$ is the SVD of M, we can write $M^T M$ as $V\Sigma^2 V^T$. Then we see that:

$$\begin{aligned}(V\Sigma^2V^T)^{\frac{1}{2}}(Dh)_M(N) + (Dh)_M(N)(V\Sigma^2V^T)^{\frac{1}{2}} &= N\\ V\Sigma V^T(Dh)_{M^TM}(N) + (Dh)_{M^TM}(N)V\Sigma V^T &= N\\ \Sigma V^T(Dh)_{M^TM}(N)V + V^T(Dh)_{M^TM}(N)V\Sigma &= V^TNV \qquad (2)\end{aligned}$$

The following proposition will help us solve this particular equation.

Proposition 4. *Let $B \in M_3$ and let D be a diagonal matrix such that $\forall i, j \in \{1,2,3\} : d_i + d_j \neq 0$. Here $d_i = D_{(i,i)}$. Then the solution to the equation:*

$$DX + XD = B$$

is given by $X = Q \bullet B$, where $Q_{i,j} = \frac{1}{d_i + d_j}$. Here $\bullet$ denotes the element-wise product of two matrices, also known as the Hadamard product.

Proof. The proof is a straightforward calculation. We have that $B_{i,j} = d_i X_{i,j} + d_j X_{i,j}$. Solving for $X_{i,j}$ we find that $X_{i,j} = \frac{B_{i,j}}{d_i + d_j}$, or in matrix notation:

$$X = Q \bullet B$$

□

Thus we see that applying Proposition 4 to equation (2) results in:

$$\begin{aligned} V^T (Dh)_{M^T M}(N) V &= Q \bullet (V^T N V) \\ (Dh)_{M^T M}(N) &= V (Q \bullet (V^T N V) V^T \end{aligned} \tag{3}$$

Finally for the derivative of the projection we have the following.

Proposition 5. *Let P be the projection as defined in equation (1). Then the derivative of P is given by:*

$$(DP)_M(N) = U(Q \bullet (U^T N V - V^T N^T U))V^T$$

Where $M = U \Sigma V^T$, and $Q_{i,j} = \frac{1}{\sigma_i + \sigma_j}$. Here σ_i is the ith singular value of M, i.e. $\sigma_i = \Sigma_{i,i}$.

Proof. We know that $P(M) = M(M^T M)^{-\frac{1}{2}}$. Let $M, N \in \mathcal{M}_3$, and $M = U \Sigma V^T$ be the SVD of M. Then:

$$\begin{aligned} (DP)_M(N) &= (D[S])_M(N)(M^T M)^{-\frac{1}{2}} + M(D[(S^T S)^{-\frac{1}{2}}])_M(N) \\ &= N(M^T M)^{-\frac{1}{2}} + M(D[(S^T S)^{-\frac{1}{2}}])_M(N) \\ &= N V \Sigma^{-1} V^T + M(D[(S^T S)^{-\frac{1}{2}}])_M(N) \end{aligned} \tag{4}$$

$$\tag{5}$$

We now separately look at $(D[(S^T S)^{-\frac{1}{2}}])_M(N)$. We first use the chain rule to rewrite it:

$$(D[(S^T S)^{-\frac{1}{2}}])_M(N) = (D[S^{-1}])_{(M^T M)^{\frac{1}{2}}} \left((D[S^{\frac{1}{2}}])_{M^T M}((D[S^T S])_M(N)) \right)$$

We can now use Propositions 1 and 2, and equation (3) to further rewrite the above derivatives:

$$\begin{aligned} &= (D[S^{-1}])_{(M^T M)^{\frac{1}{2}}} \left((D[S^{\frac{1}{2}}])_{M^T M}(N^T M + M^T N) \right) \\ &= (D[S^{-1}])_{(M^T M)^{\frac{1}{2}}} \left(V(Q \bullet (V^T (N^T M + M^T N) V)) V^T \right) \\ &= -(M^T M)^{-\frac{1}{2}} \Big(V(Q \bullet (V^T (N^T M + M^T N) V)) V^T \Big) (M^T M)^{-\frac{1}{2}} \\ &= -V \Sigma^{-1} V^T \Big(V \big(Q \bullet (V^T (N^T U \Sigma V^T + V \Sigma U^T N) V) \big) V^T \Big) V \Sigma^{-1} V^T \\ &= -V \Sigma^{-1} \big(Q \bullet (V^T N^T U \Sigma + \Sigma U^T N V) \big) \Sigma^{-1} V^T \end{aligned}$$

We now substitute this result back in equation (4):

$$\begin{aligned}(DP)_M(N) &= NV\Sigma^{-1}V^T + M(D[(S^TS)^{-\frac{1}{2}}])_M(N) \\ &= NV\Sigma^{-1}V^T - U\Sigma V^TV\Sigma^{-1}(Q \bullet (V^TN^TU\Sigma + \Sigma U^TNV))\Sigma^{-1}V^T \\ &= NV\Sigma^{-1}V^T - U(Q \bullet (V^TN^TU\Sigma + \Sigma U^TNV))\Sigma^{-1}V^T\end{aligned}$$

We now rewrite this expression to get rid of the Σ and Σ^{-1}, which will result in the desired form. Notice that here $Q^\bullet = \frac{1}{Q_{i,j}}$, i.e. element-wise inversion. Continuing from the last line:

$$\begin{aligned}&= U\big(U^TNV\Sigma^{-1}V^T - (Q \bullet (V^TN^TU + \Sigma U^TNV\Sigma^{-1}))V^T\big) \\ &= U\big(Q \bullet (Q^\bullet \bullet U^TNV\Sigma^{-1} - V^TN^TU - \Sigma U^TNV\Sigma^{-1})\big)V^T \\ &= U\big(Q \bullet (U^TNV + \Sigma U^TNV\Sigma^{-1} - V^TN^TU - \Sigma U^TNV\Sigma^{-1})\big)V^T \\ &= U(Q \bullet (U^TNV - V^TN^TU))V^T\end{aligned}$$

Where we have used the equality $Q^{-1} \bullet (A\Sigma^{-1}) = A + \Sigma A\Sigma^{-1}$, for $A \in \mathcal{M}_3$, to get to the second to last line. □

2.2.4 Projecting on the $SO(3)$

In the previous sections we have seen the shortest distance projection on the $O(3)$, and several properties of it. However, as mentioned before we are mainly interested in the $SO(3)$.

In the remainder of this paper we use the following function to project on the $SO(3)$:

$$\hat{P}(M) = \det\big(P(M)\big)P(M), \tag{6}$$

this is not the shortest distance projection on the $SO(3)$, but instead we use equation (1), and correct for the determinant. Let $M \in GL(3)$, then if $\det(M) > 0$, we have that $\hat{P} = P$. If $\det(M) < 0$, P projects M on $O(3) - SO(3)$, and so we multiply with a minus sign to get to $SO(3)$.

This fact is also important when we consider the derivative of $\hat{P}$. If $\hat{P} : A \subset GL(3)^+ \to SO(3)$, then $\hat{P} = P$, and so the derivative will be the same. If $\hat{P} : B \subset GL(3)^- \to SO(3)$, then $\hat{P} = -P$, and so the derivative will also be multiplied by a minus sign.

The actual shortest distance projection P_s on the $SO(3)$, for $M \in \mathcal{M}_3$ and $\det(M) < 0$, is as follows:

$$P_s(M) = UDV^T$$

where again $M = U\Sigma V^T$ is the SVD of M, and where $D = \text{diag}\,(1, 1, -1)$ is a diagonal matrix with 1s on the diagonal, expect for the last diagonal element, which is a -1. See [5].

2.2.5 $\mathbb{RP}^3$ and $SO(3)$

Another property of $SO(3)$ is that it is homeomorphic to $\mathbb{RP}^3$, the three dimensional real projective space. The real projective space of dimension n is formed by taking elements of $\mathbb{R}^{n+1} \setminus \{0\}$ and identifying each point by the equivalence relation $x \sim \lambda x$ for every $\lambda \in \mathbb{R} \setminus \{0\}$. An equivalent way of thinking about $\mathbb{RP}^3$ is as the ball D^3, where the antipodal points of the boundary, S^2, are identified, see [8] Example 0.4 and Section 3.D. This fact also lets us visualize $SO(3)$, which we will do later.

Consider the function $\phi : D^3 \to SO(3)$, which sends a nonzero vector x to the rotation with angle $|x|\pi$, with axis the vector through 0 and x. Using the right-hand rule we can make this rotation unambiguous. By continuity we have that ϕ sends the 0 vector to the identity. Moreover antipodal points are sent to the same rotation in $SO(3)$. Thus ϕ induces a map $\bar{\phi} : \mathbb{RP}^3 \to SO(3)$. This is a homeomorphism between the two spaces.

3 Variational auto-encoder

Now that we have discussed the preliminaries, we will explain the variational auto-encoder in more detail. After this we will explain how it was changed to form the ΔVAE , and highlight the difference using a small example.

Consider a data set $X = \{x^{(i)}\}_{i=1}^{N}$, where the samples are i.i.d. We assume this data is generated by some random process, using some unobserved random variable $\mathbf{z}$, coming from a latent space Z. This $\mathbf{z}$ can be thought of as a latent variable. The data is generated in the following way: a value $\mathbf{z}^{(i)}$ is drawn from the prior distribution $\mathbb{P}_{\theta^*}(\mathbf{z})$. Next a value $x^{(i)}$ is generated from some conditional distribution $\mathbb{P}_{\theta^*}(x|\mathbf{z})$. We assume that these distributions come from some parametric families of distributions $\mathbb{P}_\theta(\mathbf{z})$ and $\mathbb{P}_\theta(x|\mathbf{z})$. Furthermore, we introduce a recognition model, a probability distribution, $q_\phi(\mathbf{z}|x)$, which serves as an approximation to the intractable true posterior distribution $\mathbb{P}_\theta(\mathbf{z}|x)$. We will leave out the parameters for these distributions in order to avoid over cluttering.

We can see this recognition model as a probabilistic encoder, which given a data point x produces a distribution over all values of $\mathbf{z}$. Similarly we can see $\mathbb{P}(x|\mathbf{z})$ as a probabilistic decoder, which given some latent variable gives a probability distribution over the possible outcomes. Most commonly these distributions are chosen to be Gaussian distributions. This encoder will try to learn a 'good' representation of the data, and the decoder will try to generate 'good' data from these encodings. But how can we train these models to do a good job in replicating these probabilities?

3.1 The objective

What we wish to maximize is the logarithm of the marginal likelihood:

$$\log \mathbb{P}(x^{(1)}, \cdots, x^{(N)}) = \sum_{i=1}^{N} \log \mathbb{P}(x^{(i)})$$

Here we marginalize out of the latent variable $\mathbf{z}$:

$$\mathbb{P}(x^{(i)}) = \int \mathbb{P}(\mathbf{z})\, \mathbb{P}\left(x|\mathbf{z}\right) d\mathbf{z}$$

We see here that finding a better probabilistic decoder, i.e. one that is likely to produce samples from our data set, means that our objective increases. Approximately computing this integral is relatively straightforward. We can sample a large number of values $\{z_1, \cdots, z_n\}$ from $\mathbf{z}$, then $\mathbb{P}(x^{(i)}) \approx \frac{1}{n} \sum_{i=1}^{n} \mathbb{P}(x^{(i)}|z_i)$. However, this estimator is very inefficient and for practical purposes not usable.

We now first relate the recognition model and $\mathbb{P}(x^{(i)})$. We do this by means of the *Kullback-Leibler* (KL) divergence. It is a measure of how close two distributions are. The KL divergence between $q(\mathbf{z}|x)$ and $\mathbb{P}(\mathbf{z}|x)$ is given by:

$$\mathrm{KL}\left(q(\mathbf{z}|x^{(i)})||\, \mathbb{P}(\mathbf{z}|x^{(i)})\right) = \mathop{\mathbb{E}}_{q(\mathbf{z}|x^{(i)})}\left[\log q(\mathbf{z}|x^{(i)}) - \log \mathbb{P}(z|x^{(i)})\right]$$

We can then apply Bayes rule to $\mathbb{P}(\mathbf{z}|x^{(i)})$ to get:

$$\mathrm{KL}\left(q(\mathbf{z}|x^{(i)})||\, \mathbb{P}(\mathbf{z}|x^{(i)})\right) = \mathop{\mathbb{E}}_{q(\mathbf{z}|x^{(i)})}\left[\log q(\mathbf{z}|x^{(i)}) - \log \mathbb{P}(x^{(i)}|\mathbf{z}) - \log \mathbb{P}(z)\right] + \log \mathbb{P}(x^{(i)})$$

By rearranging terms this expression can be rewritten to:

$$\log \mathbb{P}(x^{(i)}) - \mathrm{KL}\left(q(\mathbf{z}|x^{(i)})||\, \mathbb{P}(\mathbf{z}|x^{(i)})\right) = \mathop{\mathbb{E}}_{q(\mathbf{z}|x)}\left[-\log q(\mathbf{z}|x^{(i)}) + \log \mathbb{P}(\mathbf{z})\right] \tag{7}$$

The KL divergence is always non-negative, so we have a lower bound, called the *evidence lower bound* or ELBO, denoted by $\mathcal{L}$, for the marginal likelihood:

$$\log \mathbb{P}(x^{(i)}) \geq \mathcal{L}(x^{(i)}) = \mathop{\mathbb{E}}_{q(\mathbf{z}|x^{(i)})} \Big[\log \mathbb{P}(x^{(i)}|\mathbf{z})\Big] - \mathrm{KL}\left(q(\mathbf{z}|x^{(i)})||\, \mathbb{P}(\mathbf{z})\right) \tag{8}$$

Now if the recognition model is a good approximation of the true posterior, then the KL term in the right-hand side of Equation 7 will be small, and so we will be almost directly optimizing $\log \mathbb{P}(x^{(i)})$ when optimizing this bound.

We want to apply gradient descent to $\mathcal{L}$ in order to optimize it. However, the gradient with respect to ϕ (the parameters for q) poses a problem. The usual estimator for gradients of the form

$$\nabla_\phi \mathop{\mathbb{E}}_{q(\mathbf{z}|x)} [f(\mathbf{z})]$$

exhibits high variance, and is impractical for use in this case, see [12] Section 2.2. Instead, under some mild conditions on q, we can rewrite this expression which yields a better approximation. This has been called the *reparametrization trick*, see [12] Section 2.4.

The reparametrization trick comes down to rewriting a random variable $\mathbf{z} \sim q_\phi(\mathbf{z}|x)$ as $\mathbf{z} = g_\phi(\epsilon, x)$, where ϵ is an auxiliary "noise" variable coming from some distribution $p(\epsilon)$, and g is some differentiable function. As an example, consider $\mathbf{z} \sim \mathcal{N}(\mu, \sigma^2)$. A valid choice for g would be $g(\epsilon) = \mu + \sigma\epsilon$, where $\epsilon \sim \mathcal{N}(0, 1)$. We then have that:

$$\nabla_\phi \mathop{\mathbb{E}}_{q(\mathbf{z}|x)} [f(\mathbf{z})] = \nabla_\phi \mathop{\mathbb{E}}_{p(\epsilon)} [f(g(\epsilon, x))] = \mathop{\mathbb{E}}_{p(\epsilon)} [\nabla_\phi f(g(\epsilon, x))] \approx \frac{1}{L} \sum_{l=1}^{L} \nabla_\phi f(g(\epsilon^{(l)}, x))$$

The KL term in $\mathcal{L}$ can be computed analytically in the case of Gaussian distributions for $\mathbb{P}(\mathbf{Z})$ and $q(\mathbf{z}|x)$. The full estimator for the ELBO, of which we can take the gradient, then becomes ([12] equation (7)):

$$\mathcal{L}(x^{(i)}) \approx \frac{1}{L} \sum_{l=1}^{L} \left(\log \mathbb{P}(x^{(i)}|g(\epsilon^{(i,l)}, x^{(i)}))\right) - \mathrm{KL}\left(q(\mathbf{z}|x^{(i)})||\, \mathbb{P}(\mathbf{z})\right)$$

3.2 Diffusion variational auto-encoders

Now that we have discussed the variational auto-encoder, we briefly explain how it was adapted into the ΔVAE . For a more detailed explanation about how this was done, see [16]. After this we highlight the difference between the ΔVAE and a standard VAE by means of an example.

3.2.1 Adapting the VAE

Notice that the latent space of the VAE is essentially $\mathbb{R}^d$, for some $d \in \mathbb{N}$. In the case where we have Gaussian distributions for the encoder q, the mean can be placed in any point in $\mathbb{R}^d$. However, in order to be able to capture topological information about the latent variables, we want to be able to restrict this behaviour, and force the latent variables to lie in a manifold of choice. The ΔVAE achieves this by changing the distributions for the encoder and the prior, such that the latent space becomes an arbitrary Riemannian manifold.

The encoder distributions are now transition probability measures of Brownian motion on the latent space Z, which is some Riemannian manifold of choice. In [16] Section 3.2, a summary of Brownian motion on manifolds is given. The encoder distribution, now denoted by $\mathbb{Q}_Z^{t,z}$, applied to some set $A \subset Z$ measures the probability that Brownian motion, started at z, at time t is in the set A. Note that z and t here have a similar role as the mean and variance, respectively, in the Gaussian distribution.

We again reparametrize this distribution, for the same reasons mentioned above. Instead of an exact reparametrization, and approximate reparametrization is constructed, see [16] Section 3.6 for details. In order to approximate Brownian motion, starting from some point z, a random step

in the ambient space is set. Then this point is projected back on the manifold. This is repeated for N steps. The reparametrization function $g : \epsilon^N \times (0, \infty) \times Z \to Z$ is then given by:

$$g(\epsilon_1, \cdots, \epsilon_N, t, z) = P\left(\cdots P\left(P\left(z + \sqrt{\frac{t}{N}}\epsilon_1\right) + \sqrt{\frac{t}{N}}\epsilon_2\right)\cdots + \sqrt{\frac{t}{N}}\epsilon_N\right)$$

For the choice of prior $\mathbb{P}(\mathbf{z})$, the uniform distribution on the manifold is chosen. This is the normalized standard measure on the manifold.

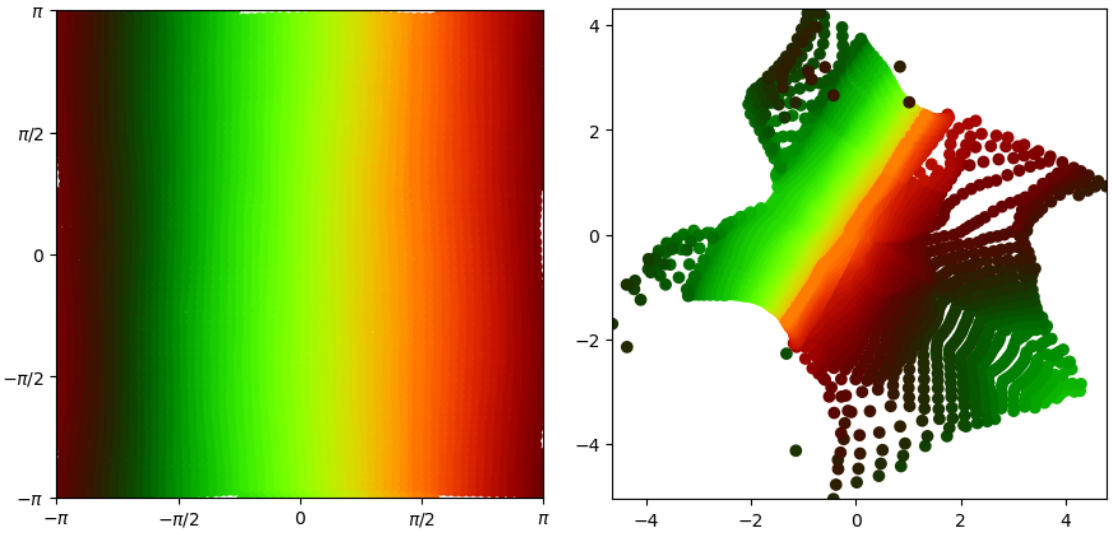
In the ΔVAE , the KL term in equation (8) cannot be computed exactly anymore. This was solved by deriving an asymptotic approximation for this term, see [16] Section 3.7. In order to ensure that this approximation remains accurate, the learned time variable t is restricted, to ensure that it does not get too large.

3.2.2 Difference between the ΔVAE and the VAE

In order to demonstrate the difference between the ΔVAE and a standard VAE, we train both of them on a particular data set. This data set consists of translations of a single picture, were the edges wrap around. It has a toroidal latent structure, by construction. See also [16] Section 4.

A torus is a two dimensional object, and so we train both a ΔVAE with a (flat) torus as latent space, and a standard VAE with a two dimensional latent space. In Fig 1 we can see the visualizations of both latent spaces after training. Here the mean that the encoder has learned is plotted. The color indicates the rotation in one direction.

We see that the ΔVAE has captured the fact that the latent space was a torus, while the standard VAE is unable to capture this fact. What we mean by capturing is that the encoder is a homeomorphism between the data space and the latent space. We see that there are points with a similar color which are on opposite sides of the latent space.



(a) A ΔVAE with a flat torus as latent space. The torus is represented by a square with periodic boundaries.

(b) A standard VAE with two dimensional latent space

Figure 1: An example highlighting the difference between the ΔVAE and a standard VAE when trained on a data set with a non-Euclidean latent structure.

However, there is nothing that prevents a standard VAE to learn an embedding of a manifold. This is exactly what happens when we train a VAE with a 3 dimensional Euclidean latent space, see Figure 2. Of course, when using the ΔVAE , we cannot sample outside of the manifold of

choice, by construction. This possible with the standard VAE. Moreover, in the case of the ΔVAE , the encoder is able to learn a homeomorphism, which cannot happen with a regular VAE.

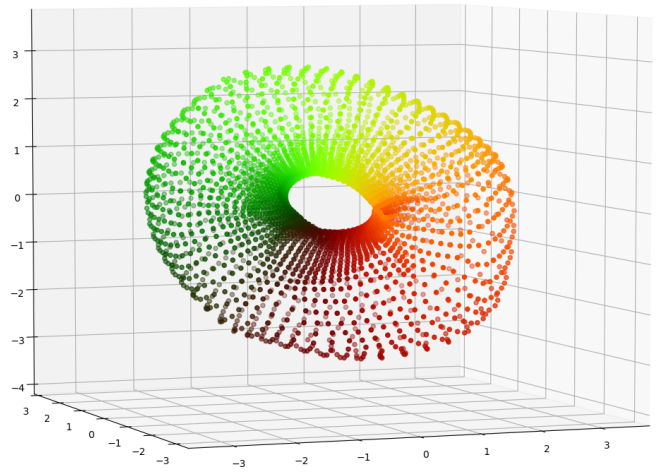


Figure 2: A standard VAE trained on the torus data set, with a three dimensional latent space. As we can see it is able to capture, to some extent, the properties of the data set.

4 Encoder functions

We would like to better understand what happens when training the ΔVAE with a $SO(3)$ latent space. We do this by looking at functions $N : SO(3) \to SO(3)$ of the form:

$$N(O) = \hat{P}(T(O)) \tag{9}$$

Where $T : \mathcal{M}_3 \to \mathcal{M}_3$ is a linear function. These resemble a linear encoder. What we mean by that is that the encoder learns for a given input a mean parameter for the encoder distribution. So we can see it as a function $SO(3) \to SO(3)$, which has exactly this form. Whenever we refer to the encoder part of the ΔVAE as a function, this is what we mean. We start by stating a topological result about functions from $\mathbb{RP}^3$ to $\mathbb{RP}^3$. Then we look at for which functions T the function N is well-defined. Next we calculate the degree of N for specific T, and present a strategy for calculating it numerically for arbitrary (well-defined) linear T.

4.1 Connection to $\mathbb{RP}^3$

As already mentioned in Section 2.2.5, the $SO(3)$ is homeomorphic to $\mathbb{RP}^3$. We will state a result concerning functions from $\mathbb{RP}^n$ to $\mathbb{RP}^n$. Of particular interest are these self functions up to homotopy, i.e. in the equivalence classes of the equivalence relation being homotopic. Let $[\mathbb{RP}^n, \mathbb{RP}^n]$ denote these equivalence classes. Then we have the following result.

Theorem 4.1 ([14] Theorem 2.1)**.** *For each integer $n \geq 1$,*

$$[\,\mathbb{RP}^n, \mathbb{RP}^n\,] \simeq \begin{cases} \mathbb{Z} & \textit{if } n \textit{ is odd} \\ \mathbb{Z}/2 & \textit{if } n \textit{ is even} \end{cases}$$

Moreover, this representation is faithfully represented in the degree of these functions. So for every $n \in \mathbb{Z}$ there is a map with degree n. This means for maps from/to projective spaces, degree fully determines homotopy class.

What this theorem tells us is that looking at the degree of the function tells us everything that we need to know from a topological view. Say for example we are able to show that the degree of some self map, associated with an encoder, is one, then we know that this function is homotopic to the identity function, which might tell us whether the encoder fully captured the topological properties of the data.

4.2 Well-definedness

The first issue we encounter when we want to try to calculate the degree of N, is determining which linear functions T satisfy the constraint of mapping $SO(3)$ into GL. This is necessary, as the projection we want to apply to the image of this function is not defined for singular matrices. Notice that if the image of T contains a matrix with positive determinant and one with negative determinant, it necessarily contains one with determinant zero. This is because $SO(3)$ is path connected. So, T should map to either $GL(3)^+$ or $GL(3)^-$.

To put it more concretely, we are looking for linear functions $T : SO(3) \to GL(3)$ satisfying the following:

$$\forall_{O \in SO(3)} : \det(T(O)) \neq 0$$

We will call functions that have this property *well-defined*. Having non-zero determinant is the same as having full rank, and since each orthogonal matrix has full rank we can rewrite this to:

$$\forall_{O \in SO(3)} : \rho(O) = \rho(T(O))$$

where we denote the rank of a matrix by ρ. Problems of this nature are known as *Linear Preserver Problems*. For an overview see, for example, [1]. In many of these related problems, the map T has a "standard form", namely:

$$T(O) = NOM \text{ or } T(O) = NO^T M, \tag{10}$$

where N and M are matrices of the appropriate size, sometimes with extra conditions imposed on them. In our case, this condition is sufficient if N and M have a non-zero determinant, as the determinant function is multiplicative and transposing a matrix preserves the determinant.

Unfortunately we do not think it is a necessary condition for T to have this form. In order to get a better idea about this, we create a data set consisting of 5000 matrices, uniformly distributed over $SO(3)$. If we then want to visualize what a function does to the determinant of these matrices, we apply this function to each of the matrices, and based on the determinant of the result, we color in the original input. A matrix is colored black if the determinant is close to 0 ($|\det| < 1e-3$), green if it is positive and red if it is negative. The coloring scales, so brighter colors mean a higher determinant.

Notice that every linear function from $\mathcal{M}_3$ to $\mathcal{M}_3$ can be represented by a 9×9 matrix. So we start investigating by randomly sampling a matrix, and using this as our linear function. Some results of this can be seen in Fig 3.

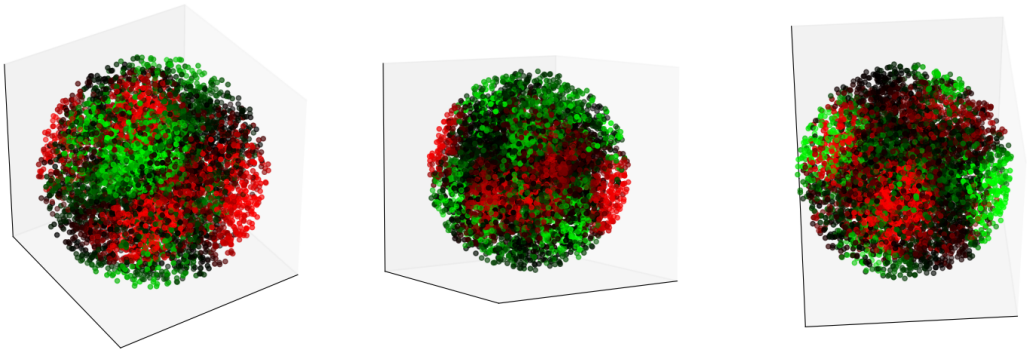
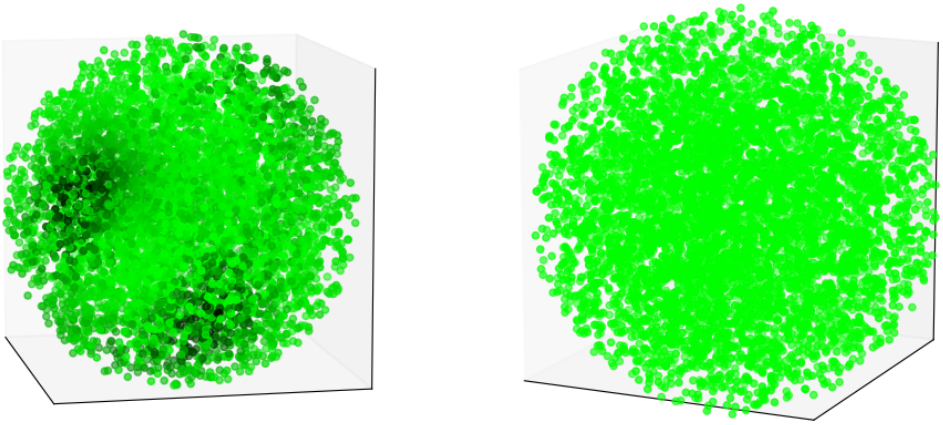


Figure 3: Three examples in which we randomly generated a 9×9 matrix and use this as our linear function.

As we can see from these examples, in general a linear function does not need to be well-defined. This is also the behaviour a linear network shows; the initial configuration of weights usually is not a well-defined one. More about that in Section 5.

What about more specific linear functions? Take for example functions of the form $T(M) = AM + MB$, where $A, B \in \mathcal{M}_3$. In general such a function cannot be rewritten to something of the form as in equation (10). These appear to behave less chaotic than the examples from Fig 3. An example can be found in Fig 4. The left picture is the original random function, and the right one was created by slightly changing some of the values in the matrices A and B.



(a) An example where we take the function $O \mapsto AO + OB$ for some random A, B. This particular example seems to be very close to mapping the entire $SO(3)$ into GL^+. We can see two smudges were the determinant gets close to 0.

(b) Playing a little more with the matrices resulted in this figure, where it seems that all $SO(3)$ matrices are mapped to GL^+.

Figure 4

We further investigate this example, as it seems the function on the right has the desired property of mapping $SO(3)$ into $G : (3)^+$. The particular function, which we will denote by T^*, is given by:

$$T^*(O) = \begin{pmatrix} 1.36 & 0.23 & 1.64 \\ 0.92 & 1.63 & 0.03 \\ 0.46 & 0.55 & 1.75 \end{pmatrix} O + O \begin{pmatrix} 0.51 & 0.53 & 0.01 \\ 0.45 & 0.38 & 0.13 \\ 0.37 & 0.59 & 0.1 \end{pmatrix}$$

Conjecture 1. *The function T^* maps $SO(3)$ into $GL(3)^+$, i.e. $T^*(SO(3)) \subset GL(3)^+$*

Unfortunately we were not able to prove this conjecture. However, we can use random sampling to get a better idea whether the proposition is true or not. Five million random $SO(3)$ matrices were generated, and the determinants of them after applying T^* were checked. This experiment suggests that $\det(T^*(O)) > 2$ for $O \in SO(3)$, meaning the proposition would be true. This would mean linear functions that are well-defined are not necessarily of the form as in (10).

4.3 The degree of left translation

We now show that if T is left translation by a fixed non-singular matrix, the degree of N is always 1. We know that when the matrix A by which we left translate has a non-zero determinant, the function T is well-defined. This tells us that there are functions of the form as in equation (9) with degree 1, and so the encoder should be able to learn such a function.

Proposition 6. *Let $A \in GL(3)$, and define $T : \mathcal{M}_3 \to \mathcal{M}_3$ by $T(M) = AM$. Then the degree of the function $N : SO(3) \to SO(3)$ given by $N(O) = \hat{P}(T(O))$ is 1. Here $\hat{P}$ is the projection on $SO(3)$, defined in equation (6).*

Proof. Let $s = \text{sign}(\det(A))$, so we have that $s = \det(P(A))$. We start by showing that N is a bijection. First let $O_1, O_2 \in SO(3)$. Then assuming $N(O_1) = N(O_2)$:

$$\begin{aligned} N(O_1) &= N(O_2) \\ \det\big(P(T(O_1))\big) \cdot P(T(O_1)) &= \det\big(P(T(O_2))\big) \cdot P(T(O_2)) \\ \det\big(P(AO_1)\big) \cdot P(AO_1) &= \det\big(P(AO_2)\big) \cdot P(AO_2) \\ s \cdot P(A)O_1 &= s \cdot P(A)O_2 \\ O_1 &= O_2 \end{aligned}$$

Where we have used Lemma 1 to take O_1 and O_2 outside of the projection, and the fact that the matrix $P(A)$ is non-singular. So, N is injective. Next let $O \in SO(3)$. Then consider $\hat{P}(A)^T O = sP(A)^T O \in SO(3)$. We have that:

$$\begin{aligned} N(\hat{P}(A)^T O) &= \hat{P}(A\hat{P}(A)^T O) \\ &= \det\Big(P\big(A\hat{P}(A)^T O\big)\Big) \cdot P\big(A\hat{P}(A)^T O\big) \\ &= \det\Big(P\big(AsP(A)^T O\big)\Big) \cdot P\big(AsP(A)^T O\big) \\ &= \det\Big(sP(A)P(A)^T O\Big) \cdot s \cdot P(A)P(A)^T O \\ &= (s)^4 O \\ &= O \end{aligned}$$

Where we again have used Lemma 1. And so we find that N is also surjective, hence bijective. Recall from Definition 7 that:

$$\deg\,(N) = \sum_{x \in N^{-1}(\hat{P}(A))} DN_x$$

We shall shortly see why $\hat{P}(A)$ is a regular value. The particular choice of $\hat{P}(A)$ for regular value is to make the following computations easier. In particular, we have that $N^{-1}(P(A)) = I$.

Recall that the orientation given to the tangent space of the $SO(3)$ at the point $\hat{P}(A)$ is given by:

$$\big(s \cdot UV^T L_x,\ s \cdot UV^T L_y,\ s \cdot UV^T L_z\big)$$

where $A = U\Sigma V^T$ is the SVD of A. Moreover, for the derivative, Proposition 5 and Section 2.2.4, we have that:

$$\begin{aligned}
(DN)_I(K) &= (D\hat{P})_{T(I)}((DT)_I(K)) \\
&= (D\hat{P})_A(AK) \\
&= s \cdot U(Q \bullet (U^T AKV - V^T K^T A^T U))V^T \\
&= s \cdot U(Q \bullet (U^T U\Sigma V^T Kv - V^T KV\Sigma U^T U))V^T \\
&= s \cdot U(Q \bullet (\Sigma V^T KV - V^T K^T V\Sigma))V^T
\end{aligned}$$

And so we have the following orientation induced on $TSO(3)_{P(A)}$ by the derivative:

$$\big((DN)_I(L_x),\ (DN)_I(L_y),\ (DN)_I(L_z)\big)$$

So now we have to check if these two orientations are the same or opposite. In order to do this we need to find a 3×3 matrix C such that:

$$(DN)_I(L_x) = C_{1,1} \cdot s \cdot UV^T L_x + C_{1,2} \cdot s \cdot UV^T L_y + C_{1,3} \cdot s \cdot UV^T L_z$$

and similarly for L_y and L_z. Thus for the first row of this matrix we need the following:

$$\begin{aligned}
s \cdot U(Q \bullet (\Sigma V^T L_x V - V^T L_x^T V\Sigma))V^T &= C_{1,1} \cdot s \cdot UV^T L_x + C_{1,2} \cdot s \cdot UV^T L_y + C_{1,3} \cdot s \cdot UV^T L_z \\
V(Q \bullet (\Sigma V^T L_x V - V^T L_x^T V\Sigma))V^T &= C_{1,1} L_x + C_{1,2} L_y + C_{1,3} L_z
\end{aligned}$$

Let $D^x := V(Q \bullet (\Sigma V^T L_x V - V^T L_x^T V\Sigma))V^T$, and similarly for D^y and D^z. We can simplify this down further:

$$\begin{aligned}
D^x &= V(Q \bullet (\Sigma V^T L_x V - V^T L_x^T V\Sigma))V^T \\
&= V(Q \bullet (\Sigma V^T L_x V + V^T L_x V\Sigma))V^T \\
&= V(V^T L_x V)V^T \\
&= L_x
\end{aligned}$$

Where we have used the fact that $Q \bullet (\Sigma A + A\Sigma) = A$ for $A \in \mathcal{M}_3$. This also holds for D^y and D^z, and so we find that C is equal to the identity matrix.

This tells us that I is a regular point, as the derivative $(DN)_I$ maps a basis into another basis. Therefore $\hat{P}(A)$ is a regular value, because N is a bijection, and because $\det(C) = 1 > 0$, we find that the degree of N is 1. □

4.4 Calculating the degree numerically

Assuming T is well-defined, we can try to numerically compute the degree using software like Mathematica. For simplicity's sake we will assume that $T : SO(3) \to GL(3)^+$. The case where T maps to $GL(3)^-$ is very similar, and only differs in a couple signs in some equations.

Let $O \in SO(3)$ be a regular value. We do essentially the same as we did with the example above, only now we calculate the inverse $N^{-1}(O)$ numerically. We can rewrite the set $N^{-1}(O)$

to a set of several polynomial equations. Let $M \in \mathcal{M}_3$, then M needs to satisfy the following if $M \in N^{-1}(O)$:

$$\det(M) = 1 \tag{11}$$
$$MM^T = I \tag{12}$$
$$O^T T(M) - T(M)^T O = 0 \tag{13}$$
$$\det(O^T T(M)_k) > 0 \ \ \forall k \in \{1, 2, 3\} \tag{14}$$

Equations (11) and (12) constrain the matrix M to be a $SO(3)$ matrix. For equations 13 and 14 we first notice the following. If we have that $N(M) = O$, then $T(M) = OS$ for some positive symmetric definite matrix S, or $O^T T(M) = S$. These two equations check for this condition, thus ensuring that the matrices we find actually map to the desired element in $SO(3)$. In equation (14) M_k means the $k \times k$ upper left sub matrix of M. This condition is equivalent to being positive symmetric definite, and is known as *Sylvester's criterion*.

Let M be a matrix we found using the above procedure. For every such matrix we need to do the following. First of all we have that $DN_M(K) : TSO(3)_M \to TSO(3)_O$ is given by:

$$(DN)_M(K) = (DP)_{T(M)}((DT)_M(K)) = (DP)_{T(M)}(T(K))$$

Notice that $DT_M(K) = T(K)$, as T is linear. The orientation of $TSO(3)_M$ is given by:

$$\big(ML_x,\ ML_y,\ ML_z\big)$$

And so we have that the orientation induced by the derivative on $TSO(3)_O$ is:

$$\big((DN)_M(ML_x),\ (DN)_M(ML_y),\ (DN)_M(ML_z)\big)$$

So we want to compare this orientation and the standard one. This amounts to finding a matrix C such that:

$$(DN)_M(ML_x) = C_{1,1}OL_x + C_{1,2}OL_y + C_{1,3}OL_z$$
$$O^T(DN)_M(ML_x) = C_{1,1}L_x + C_{1,2}L_y + C_{1,3}L_z$$

And similarly for $O^T(DN)_M(ML_y)$ and $O^T(DN)_M(ML_y)$. Now define $D^x := O^T(DN)_M(ML_x)$, and similarly for y, z. The matrix C can then be read of as follows:

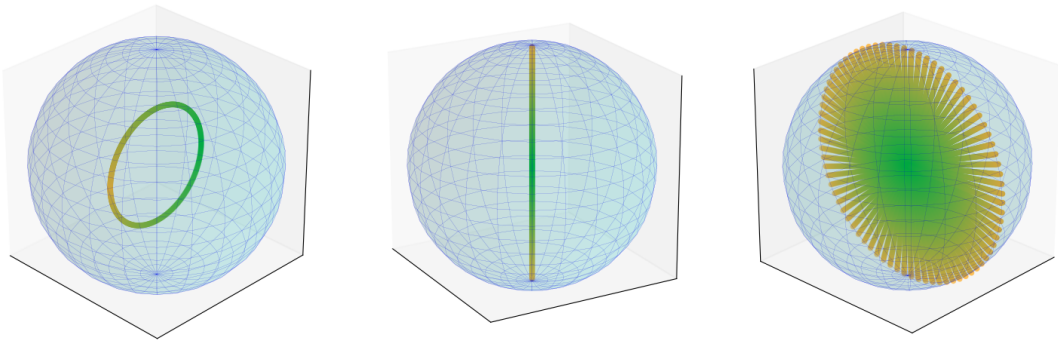
$$C = \begin{pmatrix} D^x_{3,2} & D^x_{1,3} & D^x_{2,1} \\ D^y_{3,2} & D^y_{1,3} & D^y_{2,1} \\ D^z_{3,2} & D^z_{1,3} & D^z_{2,1} \end{pmatrix}$$

and its determinant tells us if the derivative preserves or reverses orientation. Repeating this procedure for every $M \in N^{-1}(O)$, and summing the resulting sign of the determinant of C will tell us the degree of N.

5 Experiments

We have seen that for specific linear functions, the degree of the encoder is 1, and so there is a homotopy between it and the identity function. We will start with training a ΔVAE with the $SO(3)$ as latent space on the $SO(3)$ data set, a generated data set consisting of 6400 special orthogonal matrices, uniformly sampled from $SO(3)$. What we would expect is that the network learns a function that is homotopic to the identity function. The previous section showed that this should be possible. After this we train a ΔVAE on the cubes data set, consisting of 8000 images of randomly rotated cubes. This data set is generated by applying randomly sampled rotation matrices to the vertices of a cube, and drawing these to form the image.

We can visualize the latent space by plotting the mean the encoder learns for a data point. Besides visualizing the latent space, we can also create visualizations of loops and a surface inside of $SO(3)$. This is so we can better visualize what the encoder is doing when training on the $SO(3)$ data set. Fig 5 shows the two loops and the surface as they are positioned in the domain. Notice that one of the loops is a contractible loop, and one is a non-contractible one.



(a) A contractible loop in $SO(3)$

(b) A non-contractible loop in $SO(3)$.

(c) A surface in $SO(3)$.

Figure 5: Three different subsets of $SO(3)$. Notice that the two loops are not homotopic to each other. Furthermore the outline of the sphere is drawn to better understand how the subsets are positioned inside of the $SO(3)$.

5.1 The $SO(3)$ data set

5.1.1 Linear network

We will start with a ΔVAE in which the encoder part is linear, i.e. the activation functions are linear. So now the encoder looks like the function N, described in Section 4. For details about the architecture and training see Appendix A. In Table 1 several metrics can be found, also for the other architectures trained on the $SO(3)$ data set.

Architecture	**ELBO**	**KL**	**RL**	**MSE**
Linear	15.6381	6.5775	9.0606	0.1756
Linear (pre-train)	15.6028	6.5775	9.0253	0.1677
Selu	15.6232	6.5775	9.0458	0.1723
Selu (pre-train)	15.6052	6.5774	9.0277	0.1683

Table 1: Results for the $SO(3)$ data set. All metrics are computed from 10 runs. The metrics are the evidence lower bound (ELBO), KL-divergence (KL) and the reconstruction loss (RL).

These linear networks are sometimes able to capture topological properties. In Fig 6 the latent space for a particular example is visualized, in which the network managed to learn an homeomorphism. In particular, using the method described in 4.4, we can calculate the degree of the encoder part. It turns out this particular example has degree 1. So we know that the encoder part of the network learned a homeomorphism, and in this case one that is homotopic to the identity function. There have been examples were the encoder had degree -1, in which case the the learned encoder function is not homotopic to the identity anymore.

When a linear network is able to capture the properties of the $SO(3)$ data set, this is also clear from the reconstruction loss and KL loss, which are slightly lower than the averages found in Table 1.

In Fig 6 we can also see how the various subsets of $SO(3)$ changed by training. It appears they are slightly distorted, however, they are still homotopic to the original loops, which is what we would expect.

In the cases where the encoder is unable to capture the properties of the data set, the latent space looks similar to the one in Figure 7. It seems that in these cases, the encoder is unable to learn a well-defined encoder function.

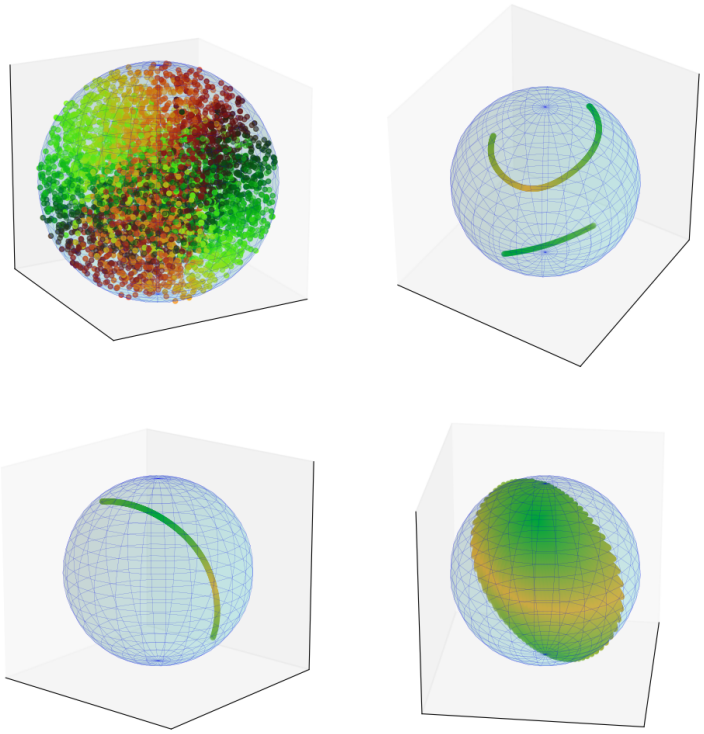


Figure 6: The top left picture shows a visualization of the latent space. The three remaining pictures show the three subsets of $SO(3)$ after training. The top right picture is the contractible loop, and the bottom left one is the non-contractible loop. The colors in the latent space visualization are based on the first Euler angle of the encoded matrix.

5.1.2 Non-linear network

We now move away from the linear networks. Unfortunately these non-linear networks seem to not be able to capture the latent structure as the linear network was sometimes able to. Despite the metrics in Table 1 being about equal, the latent space always looks messy. Fig 7 depicts a typical example of what a latent space looks like.

This messy latent space again comes down to the fact that the encoder does not seem to be able to become well-defined. In order to try to force it to learn a well-defined function, several different losses and regularizes were added to the network. Most of these placed a penalty on the layer before projecting on the manifold, where we tried to get rid of singular matrices. These are described in detail in Appendix A.3. Unfortunately, these did not seem to help the ΔVAE . Fig 8 shows what this encoder does to the determinant, after training. It seems that it gets stuck on a function like this, as the latent space looks almost identical after epoch 50.

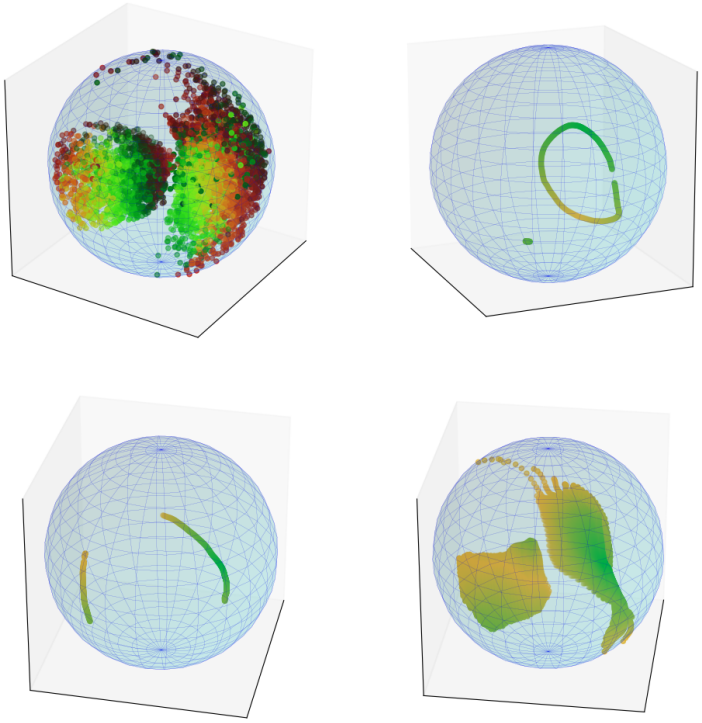


Figure 7: The top left picture shows the latent space after training. As we can see there is no real structure in it. Furthermore in the top right corner we see the contractible loop, which appears almost intact. In the bottom left corner we see the non-contractible loop, which is not intact anymore. In the bottom right the surface is depicted.

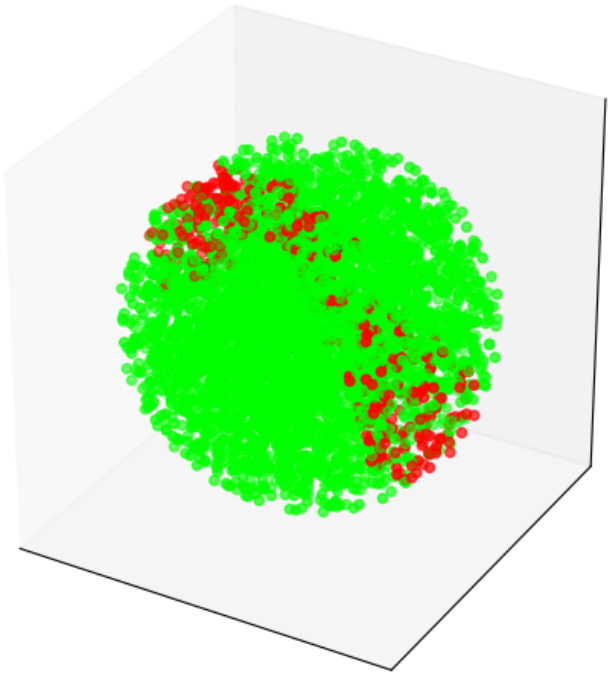
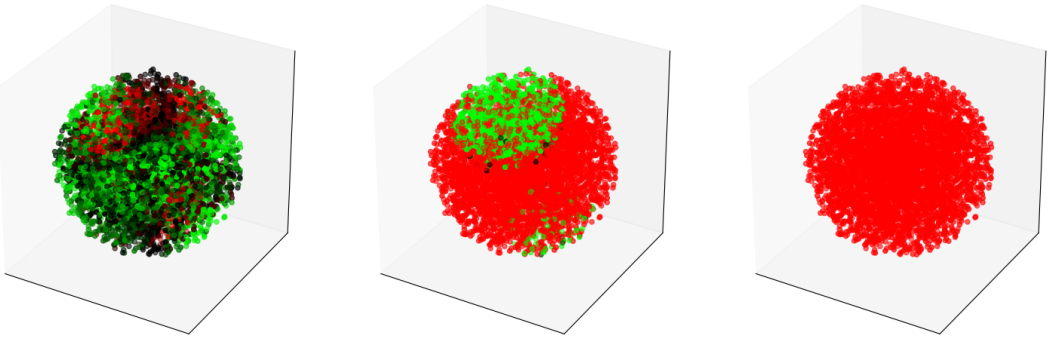


Figure 8: How the encoder function changes the determinant, after being trained for 80 epochs. This is how most encoder functions end up looking like, and it shows that the encoder is not well-defined.

5.1.3 Adapting the network

One thing that increased the performance of the ΔVAE was "pre-training" the encoder network. Before starting to train the ΔVAE , we first train the encoder separately for around 5 epochs, details can be found in Appendix A. This resulted in an encoder network that was well-defined. After this, the entire network could be trained like usual. This procedure seems to greatly increase the capabilities of the ΔVAE to learn the latent structure of the $SO(3)$ data set. Unfortunately we now do use extra information that in an unsupervised environment would not be available. Figure 9 shows visualizations of how the encoder changes the determinant, similar to how in Section 4.2 this was visualized. We can see that it learns a well-defined function very fast, in only 3 epochs. In Figure 10 we can see visualizations of the latent space and loops of a non-linear network, whose encoder is pre-trained.

As we can see, even a non-linear network is now able to capture the topological properties of the data set. The linear networks seem to be able to do this every time when using this training, and the non-linear networks succeed about half of the time. Furthermore, it seems that the number of epochs necessary for the network to be done learning is also greatly reduced.



(a) Before training. (b) After one epoch. (c) After two epochs.

Figure 9: Visualizing the encoder in the way described in Section 4.2. We see that by separately training the encoder it learns a well-defined function very fast.

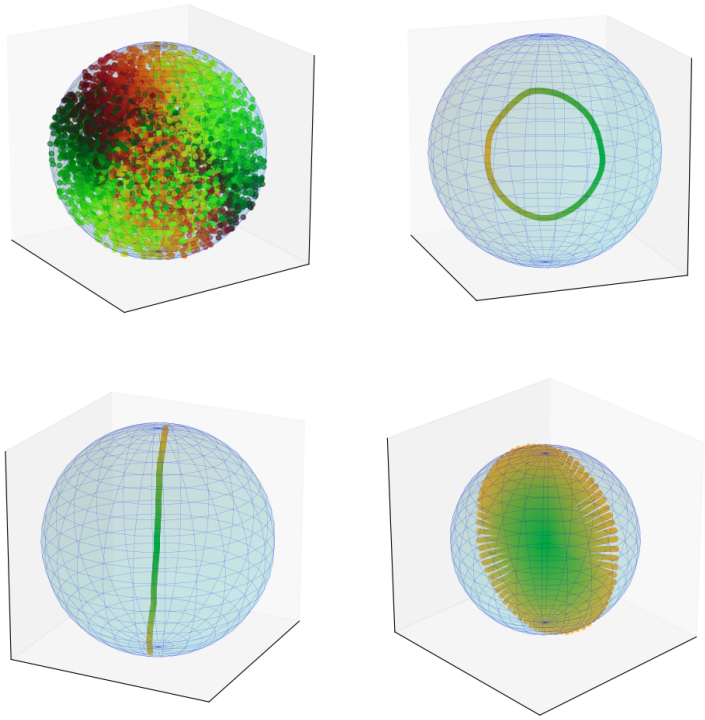


Figure 10: The top left picture again shows the latent space, which is uniformly covered in the way we would expect it to be. In the top right corner we see the contractible loop, which appears to still be contractible. This also holds for the non-contractible loop, in the bottom left corner. The surface does not appear to be altered.

5.2 Cube data set

Lastly we train the ΔVAE on a more real-life data set with a $SO(3)$ latent structure. This data set consists of pictures of cubes randomly rotated according to some element of the $SO(3)$. The particular rotation for each picture is known, so we can try to apply the training technique described in Section 5.1.3.

Architecture	**ELBO**	**KL**	**RL**	**MSE**
Linear	4547.405	7.743	4539.662	0.042
Linear (pre-train)	4519.644	7.353	4512.291	0.030
Selu	4531.442	7.701	4523.741	0.035
Selu (pre-train)	4506.269	6.577	4499.691	0.025

Table 2: Results for the cube data set. All metrics are computed from 10 runs.The metrics are the evidence lower bound (ELBO), KL-divergence (KL) and the reconstruction loss (RL).

The results of this can be seen in Table 2. As we can see, the non-linear network with pre-training performs the best. However, none of the networks were able to fully capture the $SO(3)$ latent structure, as the ΔVAE was able to do with the $SO(3)$ data set. In Fig 11 we can see a visualization of the latent space of a non-linear pre-trained network. What is immediately noticeable is the fact that it looks very unorganized. The encoder was not able to learn a homeomorphism, and moreover, if we compare it to the latent space in Fig 7 we can see that points with a similar color (which are pictures with a similar angle), are not evenly spread out.

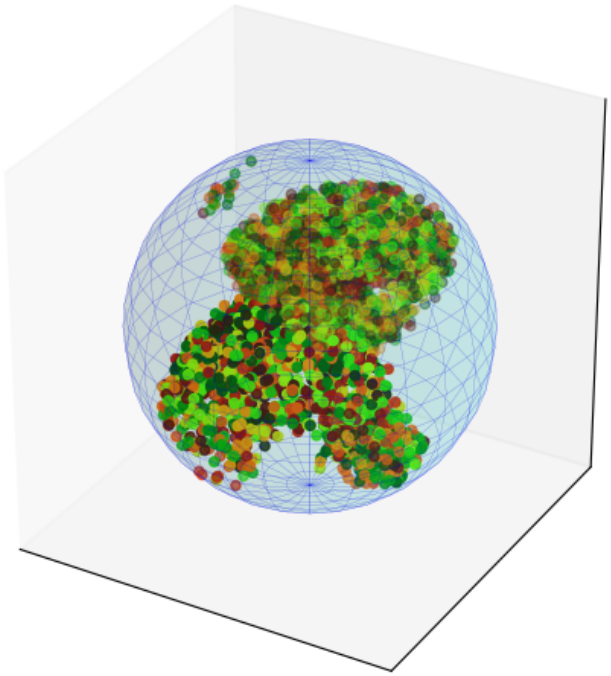


Figure 11: The latent space of the ΔVAE after training on the cube data set. We can see that it is very unorganized.

6 Conclusion

In this report we looked at the ΔVAE with the $SO(3)$ as latent space. First of all, a result about functions on $\mathbb{RP}^3$ was mentioned and its relevance for this paper explained. Next, some attention was given to the well-definedness of functions of the form $\hat{P} \circ T$, where $\hat{P}$ is the projection on $SO(3)$ (equation (6)) and T is a linear function from $SO(3)$ to $\mathcal{M}_3$. We conjectured that the functions T that are well-defined, are not of a standard form that many related problem require T to be.

After this we showed that the degree of $\hat{P} \circ T$, where T in this case is left translation by a fixed non-singular matrix, is always 1. Next to this, we outlined an approach to calculate the degree of $\hat{P} \circ T$ numerically, for any linear T.

Lastly, we trained ΔVAEs on both the $SO(3)$ data set and the cube data set, both of which have a $SO(3)$ latent space. For the $SO(3)$ data set, linear encoder networks were sometimes able to capture the topological properties of the data, however, non-linear networks were unable to do this at all. A method to improve this was discussed, which notably improved the ability of the networks to capture these properties. For the cube data set, the network was unfortunately still not able to do this.

The main problem that prevents the ΔVAE from fully capturing the properties of the data seems to be the inability of to learn a well-defined encoder function. We showed that with extra information, i.e. supervised learning, this can be resolved in the case of the $SO(3)$ data set. For the cube data set this technique seemed to improve the capabilities of the network slightly, but not fully solve the problem yet.

Further research can look at finding necessary conditions for when a linear function T is well-defined, which might reveal other, better, ways to force these encoder networks to be well-defined. These could then be applied to the encoder instead of the supervised learning which we have done now. Also, a better architecture for the network might increase the performance, especially in the case of the cube data set.

The contributions of this paper are the better understanding of the workings of the ΔVAE with a $SO(3)$ latent space. In particular, a better understanding of what the problem is that prevents the VAE from capturing topological properties of data sets with a $SO(3)$ latent structure.

A Setup

We now briefly discuss the architecture of the ΔVAE used in this paper. The Python library Tensorflow was used to make and train the network. The encoder part of the ΔVAE is a multi-layer perceptron, where we have an input layer, and several densely connected layers, from which we learn `z_mean_pro` and `z_log_t`, the parameters for the encoder distribution. To get `z_mean_pro`, We first learn a value `z_mean` and then project this on the manifold to ensure that it is actually an element of the latent space. To get `z_log_t`, we have a special layer which restricts the output of a dense layer before it. Next these two parameters are used as input for a sampling layer.

The output of this sampling is then given to the decoder part. This is again a multi-layer perceptron, consisting of several densely connected layers. The last layer has the same size as the input layer of the encoder. The loss function used by the network is the ELBO, as discussed in Section 3.

A.1 $SO(3)$ data set architecture

The networks trained on the $SO(3)$ data set all had the same architecture, which can be found in Fig 12. The boxes represent layers, where the text inside indicates what kind of layer it is. Italic text refers to the name of the layer. Furthermore, the number after the text represents the number of neurons in that layer. The output of the encoder is fed into the input of the decoder.

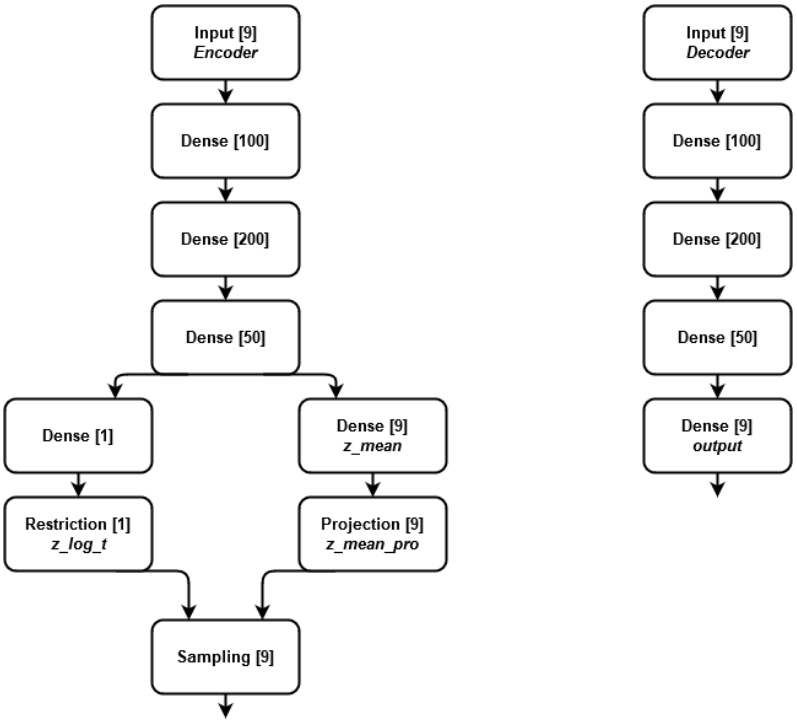


Figure 12: Architecture for the $SO(3)$ network.

The difference between the Linear and Selu is only the activation function for the encoder part of the network, the Linear having linear activations and the Selu having selu activations. The decoder always has Selu activation functions. Several different activations were tried and it appear selu worked best. None of the layers in the encoder used a bias, as it turned out this greatly improved the performance. Each network was trained for 80 epochs, using a batch size of 32.

The pre-trained networks have an encoder which was trained for 4 epochs, with as loss function

Mean Squared Error, on half of the training data. The batch size used was also 32. After this, the full network was trained again for 80 epochs.

A.2 Cube data set architecture

The networks trained on the cube data set are very similar to the ones trained on the $SO(3)$ data set. In Fig 13 the layers can be seen which were used. Again, the only difference between the Linear and Selu architecture is the activation function used by the encoder. Each network was trained for 80 epochs, using a batch size of 32.

Pre-trained networks were now trained for 8 epochs instead of 4, as it looked like the network benefited from this.

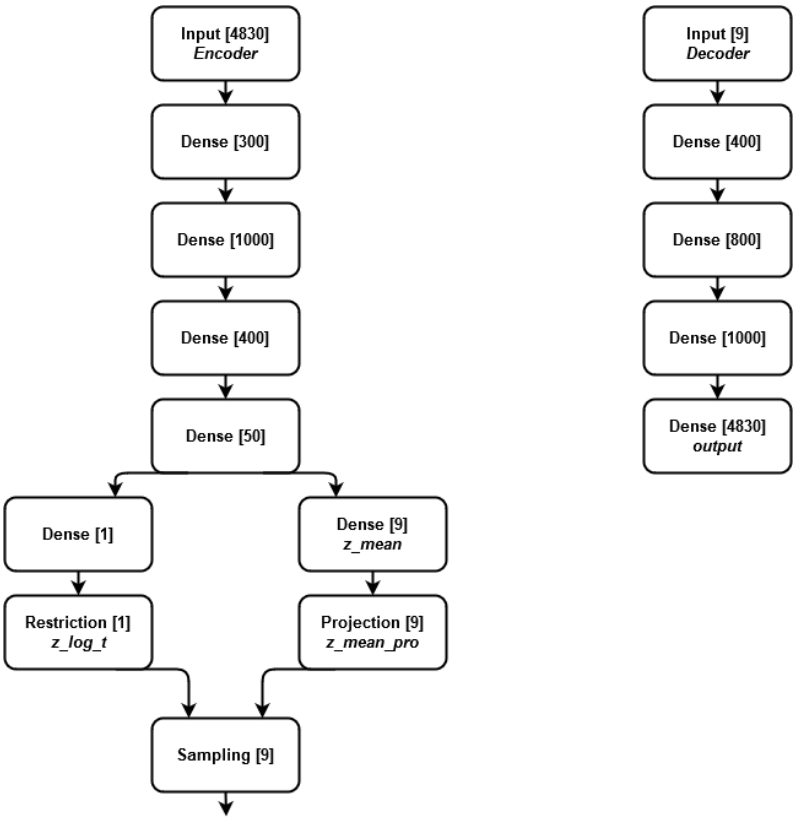


Figure 13: Architecture for the cube network.

A.3 Regularizers

We tried to add several regularizes/layers to the ΔVAE in order to help it learn a well-defined encoder. As already mentioned, these did not appear to help the ΔVAE . We discuss here what exactly was tried.

First of all we tried to add a regularizer which added a loss based upon the determinant of the input of the layer, more precisely:

$$\frac{c}{1 + \det(input)},$$

where c is a hyper parameter. This was added to the output of the `z_mean` layer. The hope was that this would encourage the ΔVAE to stay away from singular matrices.

Next, we tried adding a regularizer to constraint the trace of the output of the `z_mean` layer, because the network produces matrices with a very high trace and determinant (with or without the previously discussed regularizer).

Lastly we tried to add a layer which would always output a matrix with non-negative determinant. It would do this by calculating the determinant of the input, and multiply it with the sign of the it. This layer was added between `z_mean` and `z_mean_pro`.

References

[1] Matej Brešar, Mikhail A. Chebotar, and Wallace S. Martindale. Linear preserver problems. *Frontiers in Mathematics*, 2007(September):189–219, 2007.

[2] Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-VAE, 2018.

[3] T.R. Davidson, L. Falorsi, N. De Cao, T. Kipf, and J.M. Tomczak. Hyperspherical variational auto-encoders. In *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, volume 2, pages 856–865, 2018.

[4] Luca Falorsi, Pim de Haan, Tim R. Davidson, Nicola De Cao, Maurice Weiler, Patrick Forré, and Taco S. Cohen. Explorations in Homeomorphic Variational Auto-Encoding. 2018.

[5] J. L. Farrell, J. C. Stuelpnagel, R. H. Wessner, J. R. Velman, and J. E. Brook. A least squares estimate of satellite attitude (grace wahba). *SIAM Review*, 8(3):384–386, 1966.

[6] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. The Johns Hopkins University Press, fourth edition, 2013.

[7] Brian Hall. *Lie groups, Lie algebras, and representations : an elementary introduction*. Graduate texts in mathematics. Springer, second edition, 2015.

[8] Allen Hatcher. *Algebraic Topology*. 2001.

[9] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. β-vae: Learning basic visual concepts with a constrained variational framework. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–13, 2019.

[10] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*.

[11] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In *35th International Conference on Machine Learning, ICML 2018*, volume 6, pages 4153–4171. International Machine Learning Society (IMLS), 2018.

[12] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, (Ml):1–14, 2014.

[13] John M Lee. *Introduction to smooth manifolds*. 2013.

[14] C. A. McGibbon. Self maps of projective spaces. *Transactions of the American Mathematical Society*, 271(1):325–346, 1982.

[15] John W. Milnor. topology from the differentiable viewpoint. 1997.

[16] Luis A. Pérez Rey, Vlado Menkovski, and Jacobus W. Portegies. Diffusion Variational Autoencoders. jan 2019.

[17] Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *31st International Conference on Machine Learning, ICML 2014*, 4:3057–3070, 2014.

[18] John Stillwell. *Naive lie theory*. Undergraduate texts in mathematics. Springer, 2008.