Journal of Machine Learning Research 21 (2020) 1-28 Submitted 6/18; Revised 8/20; Published 11/20

Diffeomorphic Learning

Laurent Younes LAURENT.YOUNES@JHU.EDU
Department of Applied Mathematics and Statistics and Center for Imaging Science
Johns Hopkins University
3400 N.Charles st.
Baltimore, MD 21209

Editor: Corinna Cortes

Abstract

We introduce in this paper a learning paradigm in which training data is transformed by a diffeomorphic transformation before prediction. The learning algorithm minimizes a cost function evaluating the prediction error on the training set penalized by the distance between the diffeomorphism and the identity. The approach borrows ideas from shape analysis where diffeomorphisms are estimated for shape and image alignment, and brings them in a previously unexplored setting, estimating, in particular diffeomorphisms in much larger dimensions. After introducing the concept and describing a learning algorithm, we present diverse applications, mostly with synthetic examples, demonstrating the potential of the approach, as well as some insight on how it can be improved.

Keywords: Diffeomorphisms, Reproducing kernel Hilbert Spaces, Classification

1. Introduction

We consider, in this paper, a family of classifiers that take the form $x \mapsto F(\psi(x))$, $x \in \mathbb{R}^d$, where ψ is a diffeomorphism of $\mathbb{R}^d$ and F is real-valued or categorical, belonging to a class of simple (e.g., linear) predictors. We will describe a training algorithm that, when presented with a finite number of training samples, displaces all points together through non-linear trajectories, in order to bring them to a position for which the classes are linearly separable. Because these trajectories are built with a guarantee to avoid collisions between points, they can be interpolated to provide a fluid motion of the whole space ($\mathbb{R}^d$) resulting in a diffeomorphic transformation. One can then use this transformation to assign a class to any new observation. Because one may expect that the simplicity of the transformation will be directly related to the ability of the classifier to generalize, point trajectories and their interpolation are optimized so that this resulting global transformation is penalized from erring too far away from the identity, this penalty being assessed using a geodesic distance in the group of diffeomorphisms, based on methods previously developed for shape analysis.

The last decade's achievements in deep learning have indeed demonstrated that optimizing nonlinear transformations of the data in very high dimensions and using massive parametrization could lead to highly performing predictions without being necessarily struck by the curse of dimensionality. The approach that we describe here also explores a "very large" space in terms, at least, of the number of parameters required to describe the transformations, and uses a training algorithm that implements, like neural networks, dynamic programming. It however frames the estimated transformations within a well specified space of diffeomorphisms, whose nonlinear structure is adapted to the successive compositions of functions that drive deep learning methods. While it shares some of the characteristics of deep methods, our formulation relies on a regularization term, which makes it closer in spirit with many of the methods used for non-parametric prediction.

We now sketch our model for classification, and for this purpose introduce some notation. We let c denote the number of classes for the problem at hand. As a consequence, the function F can either take values in $C = \{0, \ldots, c-1\}$, or in the space of probability distributions on C (which will be our choice since we will use logistic regression). We furthermore assume that F is parametrized, by a parameter $\theta \in \mathbb{R}^q$, and we will write $F(x;\theta)$ instead of $F(x)$ when needed.

Assume that a training set is given, in the form $\mathcal{T}_0 = (x_1, y_1, \ldots, x_N, y_N)$ with $x_k \in \mathbb{R}^d$ and $y_k \in C$, for $k = 1, \ldots, N$. We will assume that this set is not redundant, i.e., that $x_i \neq x_j$ whenever $i \neq j$. (Our construction can be extended to redundant training sets by allowing the training class variables y_k to be multi-valued. This would result in somewhat cumbersome notation that is better avoided.) If ψ is a diffeomorphism of $\mathbb{R}^d$, we will let $\psi \cdot \mathcal{T}_0 = (\psi(x_1), y_1, \ldots, \psi(x_N), y_N)$. The learning procedure will consist in minimizing the objective function

$$G(\psi, \theta) = D(\mathrm{id}, \psi)^2 + \lambda g(\theta) + \frac{1}{\sigma^2}\Gamma(F(\cdot, \theta), \psi \cdot \mathcal{T}_0) \tag{1}$$

with respect to ψ and θ. Here D is a Riemannian distance in a group of diffeomorphisms of $\mathbb{R}^d$, that will be described in the next section, Γ is a "standard" loss function, g a regularization term on the parameters of the final classifier and λ, σ are positive numbers. One can take for example

$$\Gamma(F(\cdot, \theta), \psi \cdot \mathcal{T}_0) = -\sum_{k=1}^{N} \log F(\psi(x_k); \theta)(y_k), \tag{2}$$

where

$$F(z, \theta)(y) = \frac{e^{\theta(y)^T z}}{\sum_{y' \in C} e^{\theta(y')^T z}}$$

and where the parameter is $(\theta_1, \ldots, \theta_{c-1}) \in (\mathbb{R}^d)^{c-1}$ with $\theta_0 = 0$, while (as done in our experiments) $g(\theta) = |\theta|^2$.

The proposed construction shares some of its features with feed-forward networks (Goodfellow et al., 2016), for which, for example, using the loss function above on the final layer is standard, and can also be seen as a kernel method, sharing this property with classifiers such as support vector machines (SVMs) or other classification methods that use the "kernel trick" (Vapnik, 2013; Schölkopf and Smola, 2002). However, the algorithm is directly inspired from diffeomorphic shape analysis, and can be seen as a variant of the large deformation diffeomorphic metric mapping (LDDMM) algorithm, which has been introduced for image and shape registration (Joshi and Miller, 2000; Beg et al., 2005; Younes, 2010). Up to our knowledge, this theory has not been applied so far to classification problems prior to the development of the present work, although some similar ideas using a linearized model have been suggested in (Trouvé and Yu, 2001) for the inference of metrics in shape spaces, and a similar approach has been discussed in Walder and Schölkopf (2009) in the context of dimensionality reduction. Invertible neural networks have been introduced as "normalizing flows" in (Rezende and Mohamed, 2015), with a main focus on generative modeling and variational inference. (Normalizing flows are discrete iterates of invertible functions that are implementable within a neural architecture; see Kobyzev et al. (2020) for a recent review and discussion.) Since, or concurrently to, the original submission of the present work, several papers have proposed to use flows of ODE within a machine learning context, mostly through an adaptation of residual networks (He et al., 2016) and normalizing flows to a continuous time setting (Salman et al., 2018; Chen et al., 2019; Rousseau et al., 2019; Dupont et al., 2019; Vialard et al., 2020). Additional comments on neural ODEs are provided in section 5.2.

While the underlying theory in shape analysis, which formulates the estimation of transformations as optimal control problems, is by now well established, the present paper introduces several original contributions. Translating shape analysis methods designed for small dimensional problems to larger scale problems indeed introduces new challenges, and suggests new strategies on the design of the flows that control the diffeomorphic transformations, on the choice of parameters driving the model and on optimization schemes. Importantly, the paper provides experimental evidence that diffeomorphic methods can be competitive in machine learning contexts. Indeed, diffeomorphisms in high dimension are "wild objects," and, even with the kind of regularization that is applied in this paper, it was not obvious that using them in a non-parametric setting would avoid overfitting and compare, often favorably, with several state-of-the-art machine learning methods. As explained in section 5.1, the model (after the addition of one or more "dummy" dimensions) can represent essentially any function of the data.

In this regard, given that one is ready to work with such high-dimensional objects, the requirement that the transformation is a diffeomorphism is not a strong restriction to the scope of the model. Using such a representation has several advantages, however. It indeed expresses the model in terms of a well-understood non-linear space of functions (the group of diffeomorphisms), which has a simple algebraic structure, and a differential geometry extensively explored in the literature. The description of such models is, as a consequence, quite concise and explicit, and lends itself to an optimal control formulation with the learning algorithms that result from it. Finally, there are obvious advantages, on the generative side, in using invertible data transformations, especially when the data in the transformed configuration may be easier to describe using a simple statistical model, since, in that case, the application of the inverse diffeomorphism to realizations of that simple model provides a generative model in the original configuration space.

The paper is organized as follows. Section 2 introduces basic concepts related to groups of diffeomorphisms and their Riemannian distances. Section 3 formulates the optimal control problem associated to the estimation of the diffeomorphic predictor, introducing its reduction to a finite-dimensional parametrization using a kernel trick and describing its optimality conditions. Some additional discussion on the choice of reproducing kernel is provided in section 4. Section 5 introduces remarks and extensions that are specific to the prediction problems that we consider here, and sections 6 and 7 provide experimental results, including details on how (hyper-)parameters used in our model can be set. We conclude the paper in section 8.

2. Distance on Diffeomorphisms

We now describe the basic framework leading to the definition of Riemannian metrics on groups of diffeomorphisms. While many of the concepts described in these first sections are adapted from similar constructions in shape analysis (Younes, 2010), this short presentation helps making the paper self-contained and accessible to readers without prior knowledge of that domain.

Let $\mathbf{B}_p = C_0^p(\mathbb{R}^d, \mathbb{R}^d)$ denote the space of p-times continuously differentiable functions that tend to 0 (together with their first p derivatives) to infinity. This space is a Banach space, for the norm

$$\|v\|_{p,\infty} = \max_{0\leq k\leq p} \|d^k v\|_\infty$$

where $\|\cdot\|_\infty$ is the usual supremum norm.

Introduce a Hilbert space V of vector fields on $\mathbb{R}^d$ which is continuously embedded in $\mathbf{B}_p$ for some $p \geq 1$, which means that there exists a constant C such that

$$\|v\|_{p,\infty} \leq C\|v\|_V$$

for all v in V, where $\|\cdot\|_V$ denotes the Hilbert norm on V (and we will denote the associated inner product by $\langle \cdot\,,\,\cdot \rangle_V$). This assumption implies that V is a reproducing kernel Hilbert space (RKHS). Because V is a space of vector fields, the definition of the associated kernel slightly differs from the usual case of scalar valued functions in that the kernel is matrix valued. More precisely, a direct application of Riesz's Hilbert space representation theorem implies that there exists a function

$$K : \mathbb{R}^d \times \mathbb{R}^d \to \mathcal{M}_d(\mathbb{R})$$

where $\mathcal{M}_d(\mathbb{R})$ is the space of d by d real matrices, such that

1. The vector field $K(\cdot, y)a : x \mapsto K(x, y)a$ belongs to V for all $y, a \in \mathbb{R}^d$.
2. For $v \in V$, for all $y, a \in \mathbb{R}^d$, $\langle K(\cdot, y)a\,,\, v\rangle_V = a^T v(y)$.

These properties imply that $\langle K(\cdot, x)a\,,\, K(\cdot, y)b\rangle_V = a^T K(x, y)b$ for all $x, y, a, b \in \mathbb{R}^d$, which in turn implies that $K(y, x) = K(x, y)^T$ for all $x, y \in \mathbb{R}^d$.

Diffeomorphisms can be generated as flows of ordinary differential equations (ODEs) associated with time-dependent elements of V. More precisely, let $v \in L^2([0, 1], V)$, i.e., $v(t) \in V$ for $t \in [0, 1]$ and

$$\int_0^1 \|v(t)\|_V^2 \, dt < \infty.$$

Then, the ODE $\partial_t y = v(t, y)$ has a unique solution over $[0, 1]$, and the flow associated with this ODE is the function $\varphi^v : (t, x) \mapsto y(t)$ where y is the solution starting at x. This flow is, at all times, a diffeomorphism of $\mathbb{R}^d$, and satisfies the equation $\partial_t \varphi = v(t) \circ \varphi$, $\varphi(0) = \mathrm{id}$ (the identity map). Here, and in the rest of this paper, we make the small abuse of notation of writing $v(t)(y) = v(t, y)$, where, in this case, $v(t) \in V$ for all $t \in [0, 1]$. Similarly, we will often write $\varphi^v(t)$ for the function $x \mapsto \varphi^v(t, x)$, so that φ^v may be considered either as a time-dependent diffeomorphism, or as a function of both time and space variables.

The set of diffeomorphisms that can be generated in such a way forms a group, denoted Diff_V since it depends on V. Given $\psi_1 \in \mathrm{Diff}_V$, one defines the optimal deformation cost $\Lambda(\psi_1)$ from id to ψ_1 as the minimum of $\int_0^1 \|v(t)\|_V^2 \, dt$ over all $v \in L^2([0, 1], V)$ such that $\varphi^v(1) = \psi_1$. If we let $D(\psi_1, \psi_2) = \Lambda(\psi_2 \circ \psi_1^{-1})^{1/2}$ then D is a geodesic distance on Diff_V associated with the right-invariant Riemannian metric generated by $v \mapsto \|v\|_V$ on V. We refer to Younes (2010) for more details and additional properties on this construction. For our purposes here, we only need to notice that the minimization of the objective function in (1) can be rewritten as an optimal control problem minimizing

$$E(v, \theta) = \int_0^1 \|v(t)\|_V^2 \, dt + \lambda g(\theta) + \frac{1}{\sigma^2} \Gamma(F(\cdot, \theta), \varphi(1) \cdot \mathcal{T}_0) \tag{3}$$

over $v \in L^2([0, 1], V)$, $\theta \in \mathbb{R}^q$ and subject to the constraint that $\varphi(t)$ satisfies the equation $\partial_t \varphi = v \circ \varphi$ with $\varphi(0) = \mathrm{id}$. We point out that, under mild regularity conditions on the dependency of Γ with respect to $\mathcal{T}_0$ (continuity in $x_1, \ldots, x_N$ suffices), a minimizer of this function in v for fixed θ always exists, with $v \in L^2([0, 1], V)$.

3. Optimal Control Problem

3.1 Reduction

The minimization in (3) can be reduced using an RKHS argument, similar to the kernel trick invoked in standard kernel methods (Aronszajn, 1950; Duchon, 1977; Meinguet, 1979; Wahba, 1990; Schölkopf and Smola, 2002). Let $z_k(t) = \varphi(t, x_k)$. Because the endpoint cost Γ only depends on $(z_1(1), \ldots, z_N(1))$, it suffices to compute these trajectories, which satisfy $\partial_t z_k = v(t, z_k)$. This implies that an optimal v must be such that, at every time t, $\|v(t)\|_V^2$ is minimal over all $\|w\|_V^2$ with w satisfying $w(z_k) = v(t, z_k)$, which requires $v(t)$ to take the form

$$v(t, \cdot) = \sum_{k=1}^{N} K(\cdot, z_k(t)) a_k(t) \tag{4}$$

where K is the kernel of the RKHS V and $a_1, \ldots, a_N$ are unknown time-dependent vectors in $\mathbb{R}^d$, which provide our reduced variables. Letting $\boldsymbol{a} = (a_1, \ldots, a_N)$, the reduced problem requires to minimize

$$E(\boldsymbol{a}(\cdot), \theta) = \int_0^1 \sum_{k,l=1}^{N} a_k(t)^T K(z_k(t), z_l(t)) a_l(t) \, dt + \lambda g(\theta) + \frac{1}{\sigma} \Gamma(F(\cdot, \theta), \mathcal{T}(\boldsymbol{z}(1))) \tag{5}$$

subject to $\partial_t z_k = \sum_{l=1}^{N} K(z_k, z_l) a_l$, $z_k(0) = x_k$, with the notation $\boldsymbol{z} = (z_1, \ldots, z_N)$ and $\mathcal{T}(\boldsymbol{z}) = (z_1, y_1, \ldots, z_N, y_N)$.

3.2 Optimality Conditions and Gradient

We now consider the minimization problem with fixed θ (optimization in θ will depend on the specific choice of classifier F and risk function Γ). For the optimal control problem (5), the "state space" is the space in which the "state variable" $\boldsymbol{z} = (z_1, \ldots, z_N)$ belongs, and is therefore $Q = (\mathbb{R}^d)^N$. The control space contains the control variable $\boldsymbol{a}$, and is $U = (\mathbb{R}^d)^N$.

Optimality conditions for the variable $\boldsymbol{a}$ are provided by Pontryagin's maximum principle (PMP). They require the introduction of a third variable (co-state), denoted $\boldsymbol{p} \in Q$, and of a control-dependent Hamiltonian $H_{\boldsymbol{a}}$ defined on $Q \times Q$ given, in our case, by

$$H_{\boldsymbol{a}}(\boldsymbol{p}, \boldsymbol{z}) = \sum_{k,l=1}^{N} (p_k - a_k)^T K(z_k, z_l) a_l. \tag{6}$$

(In this expression, $\boldsymbol{a}$, $\boldsymbol{p}$ and $\boldsymbol{z}$ do not depend on time.) The PMP (Hocking, 1991; Macki and Strauss, 2012; Miller et al., 2015; Vincent and Grantham, 1997) then states that any optimal solution $\boldsymbol{a}$ must be such that there exists a time-dependent co-state satisfying

$$\begin{cases} \partial_t \boldsymbol{z} = \partial_{\boldsymbol{p}} H_{\boldsymbol{a}(t)}(\boldsymbol{p}(t), \boldsymbol{z}(t)) \\ \partial_t \boldsymbol{p} = -\partial_{\boldsymbol{z}} H_{\boldsymbol{a}(t)}(\boldsymbol{p}(t), \boldsymbol{z}(t)) \\ \boldsymbol{a}(t) = \operatorname{argmax}_{\boldsymbol{a}'} H_{\boldsymbol{a}'}(\boldsymbol{p}(t), \boldsymbol{z}(t)) \end{cases} \tag{7}$$

with boundary conditions $\boldsymbol{z}(0) = (x_1, \ldots, x_N)$ and

$$\boldsymbol{p}(1) = -\frac{1}{\sigma^2} \partial_{\boldsymbol{z}} \Gamma(F(\cdot, \theta), \mathcal{T}(\boldsymbol{z}(1))). \tag{8}$$

These expressions are closely related to those allowing for the computation of the differential of E with respect to $\boldsymbol{a}(\cdot)$, which is given by

$$\partial_{\boldsymbol{a}(\cdot)} E(\boldsymbol{a}(\cdot), \theta) = \boldsymbol{u}(\cdot)$$

with

$$u_k(t) = \sum_{l=1}^{N} K(z_k(t), z_l(t))(p_l(t) - 2a_l(t)) \tag{9}$$

where p solves

$$\begin{cases} \partial_t \boldsymbol{z} = \partial_{\boldsymbol{p}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t)) \\ \partial_t \boldsymbol{p} = -\partial_{\boldsymbol{z}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t)) \end{cases} \tag{10}$$

with boundary conditions $\boldsymbol{z}(0) = (x_1, \ldots, x_N)$ and

$$\boldsymbol{p}(1) = -\frac{1}{\sigma^2} \partial_{\boldsymbol{z}} \Gamma(F(\cdot, \theta), \mathcal{T}(\boldsymbol{z}(1))).$$

Concretely, the differential is computed by (i) solving the first equation of (10), which does not involve $\boldsymbol{p}$, (ii) using the obtained value of $\boldsymbol{z}(1)$ to compute $\boldsymbol{p}(1)$ from the boundary condition, then (iii) solving the second equation of (10) backward in time to obtain $\boldsymbol{p}$ at all times, and (iv) plugging $\boldsymbol{p}$ in the expression of $\boldsymbol{u}(t)$.

For practical purposes, the discrete-time version of the problem is obviously more useful, and its differential is obtained from a similar dynamic programming (or back-propagation) computation. Namely, discretize time over $0, 1, \ldots, T$ and consider the objective function

$$E(\boldsymbol{a}(\cdot), \theta) = \frac{1}{T} \sum_{t=0}^{T-1} \sum_{k,l=1}^{N} a_k(t)^T K(z_k(t), z_l(t)) a_l(t)\, dt + \lambda g(\theta) + \frac{1}{\sigma^2} \Gamma(F(\cdot, \theta), \mathcal{T}(\boldsymbol{z}(T))) \tag{11}$$

subject to

$$z_k(t+1) = z_k(t) + \frac{1}{T} \sum_{l=1}^{N} K(z_k(t), z_l(t)) a_l(t), \quad z_k(0) = x_k.$$

We therefore use a simple Euler scheme to discretize the state ODE. Note that the state is discretized over $0, \ldots, T$ and the control over $0, \ldots, T-1$. The differential of E is now given by the following expression, very similar to that obtained in continuous time,

$$\partial_{\boldsymbol{a}(\cdot)} E(\boldsymbol{a}(\cdot), \theta) = \boldsymbol{u}(\cdot)$$

with

$$u_k(t) = \sum_{l=1}^{N} K(z_k(t), z_l(t))(p_l(t) - 2a_l(t)), \quad t = 0, \ldots, T-1$$

where p (discretized over $0, \ldots, T-1$), can be computed using

$$\begin{cases} \boldsymbol{z}(t+1) = \boldsymbol{z}(t) + \frac{1}{T} \partial_{\boldsymbol{p}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t)) \\ \boldsymbol{p}(t-1) = \boldsymbol{p}(t) + \frac{1}{T} \partial_{\boldsymbol{z}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t)) \end{cases} \tag{12}$$

with boundary conditions $\boldsymbol{z}(0) = (x_1, \ldots, x_N)$ and

$$\boldsymbol{p}(T-1) = -\frac{1}{\sigma^2} \partial_{\boldsymbol{z}} \Gamma(F(\cdot, \theta), \mathcal{T}(\boldsymbol{z}(T))).$$

These computations allow us to compute the differential of the objective function with respect to $\boldsymbol{a}$. The differential in θ depends on the selected terminal classifier and its expression for the function chosen in (2) is standard.

We emphasize the fact that we are talking of differential of the objective function rather than its gradient. Our implementation uses a Riemannian (sometimes called "natural") gradient with respect to the metric

$$\left\langle \eta_1(\cdot)\,,\, \eta_2(\cdot) \right\rangle_{\boldsymbol{a}(\cdot)} = \int_0^1 \sum_{k,l=1}^{n} \eta_k(t)^T K(z_k(t), z_l(t)) \eta_l(t) dt$$

with $\partial_t z_k = \sum_{l=1}^n K(z_k(t), z_l(t)) a_l(t)$. With respect to this metric, one has

$$\nabla_{\boldsymbol{a}(\cdot)} E(\boldsymbol{a}(\cdot), \theta) = \boldsymbol{p} - 2\boldsymbol{a},$$

a very simple expression that can also be used in the discrete case. Using this Riemannian inner product as a conditioner for the deformation parameters (and a standard Euclidean inner product on the other parameters), experiments in sections 6 and 7 run Polak-Ribiere conjugate gradient iterations (Nocedal and Wright, 1999) to optimize the objective function. (We also experimented with limited-memory BFGS, which does not use natural gradients, and found that conditioned conjugate gradient performed better on our data.)

4. Kernel

4.1 General Principles

To fully specify the algorithm, one needs to select the RKHS V, or, equivalently, its reproducing kernel, K. They constitute important components of the model because they determine the regularity of the estimated diffeomorphisms. We recall that K is a kernel over vector fields, and therefore is matrix valued. One simple way to build such a kernel is to start with a scalar positive kernel $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and let

$$K(x, y) = \kappa(x, y)\mathrm{Id}_{\mathbb{R}^d}. \tag{13}$$

We will refer to such kernels as "scalar."

One can choose κ from the large collection of known positive kernels (and their infinite number of possible combinations; Aronszajn (1950); Dyn (1989); Schölkopf and Smola (2002); Buhmann (2003)). Most common options are Gaussian kernels,

$$\kappa(x, y) = \exp(-|x - y|^2/2\rho^2), \tag{14}$$

or Matérn kernels of class C^k,

$$\kappa(x, y) = P_k(|x - y|/\rho)\exp(-|x - y|/\rho), \tag{15}$$

where P_k is a reversed Bessel polynomial of order k. In both cases, ρ is a positive scale parameter. The Matérn kernels have the nice property that their associated RKHS is equivalent to a Sobolev space of order $k + d/2$.

Vector fields v in the RKHS associated with a matrix kernel such as (13), where κ is a radial basis function (RBF), are such that each coordinate function of v belongs to the scalar RKHS associated with κ, which is translation and rotation invariant (i.e., the transformations that associate to a scalar function h the functions $x \mapsto h(R^T(x - b))$ are isometries, for all rotation matrices R and all vectors $b \in \mathbb{R}^d$).

4.2 Graph-Based Kernels

While (13) provide a simple recipe for the definition of matrix-valued kernels, other interesting choices can be made within this class. Rotation and translation invariance more adapted to spaces of vector fields, in which one requires that replacing $v : \mathbb{R}^d \to \mathbb{R}^d$ by $x \mapsto Rv(R^T(x - b))$ is an isometry of V for all R, b, leads to a more general class of matrix kernels extensively discussed in (Micheli and Glaunès, 2014). When the data is structured, however (e.g., when it is defined over a grid), rotation invariance may not be a good requirement since it allows, for example, for permuting coordinates, which would break the data structure. In this context, other choices may be preferable, as illustrated by the following example. Assume that the data is defined over a graph, say $\mathcal{G}$ with d vertices. Then, one may consider matrix kernels relying on this structure. For example, letting $\mathcal{N}_i$ denote the set of nearest neighbors of i in $\mathcal{G}$, one can simply take

$$K(x, y) = \mathrm{diag}(\Phi(|P_i x - P_i y|), i = 1, \ldots, d) \tag{16}$$

where $P_i x$ is the vector $(x_j, j \in \mathcal{N}_i)$ and Φ is an RBF associated to a positive radial scalar kernel.

4.3 Introducing Affine Transformations

RKHS's of vector fields built from RBF's have the property that all their elements (and several of their derivatives) vanish at infinity. As a consequence, these spaces do not contain simple transformations, such as translations or more general affine transformations. It is however possible to complement them with such mappings, defining

$$\hat{V}_{\mathfrak{a}} = \{g + v : g \in \mathfrak{a}, v \in V\}$$

where $\mathfrak{a}$ is any Lie sub-algebra of the group of affine transformations (so that any element $g \in \mathfrak{a}$ takes the form $g(x) = Ax + b$, where A is a matrix and b is a vector). In particular, $\mathfrak{a}$ can be the whole space of affine transformations. Since $\mathfrak{a}$ and v intersect only at $\{0\}$, one can define without ambiguity a Hilbert norm on $\hat{V}_{\mathfrak{a}}$ by letting

$$\|g + v\|^2_{\hat{V}_{\mathfrak{a}}} = \|g\|^2_{\mathfrak{a}} + \|v\|^2_V$$

where $\|g\|_{\mathfrak{a}}$ is any inner-product norm on $\mathfrak{a}$. A simple choice, for $g(x) = Ax + b$, can be to take

$$\|g\|^2_{\mathfrak{a}} = \kappa_1 \mathrm{trace}(AA^T) + \kappa_2 |b|^2. \tag{17}$$

Both $\mathfrak{a}$ and $\hat{V}_{\mathfrak{a}}$ are RKHS's in this case, respectively with kernels $K_{\mathfrak{a}}$ and $K_{\mathfrak{a}} + K$, where K is the kernel of V. If the norm on $\mathfrak{a}$ is given by (17), then

$$K_{\mathfrak{a}}(x, y) = \left(\frac{x^T y}{\kappa_1} + \frac{1}{\kappa_2}\right) \mathrm{Id}_{\mathbb{R}^d},$$

as can be deduced from the definition of a reproducing kernel.

Instead of using this extended RKHS, another option is to model affine transformations separately from the vector field. This leads to replacing (4) by

$$\hat{v}(t, \cdot) = g(t, \cdot) + \sum_{k=1}^{N} K(\cdot, z_k(t)) a_k(t)$$

and the cost (5) by

$$E(\boldsymbol{a}(\cdot), \theta) = \int_0^1 \|g(t)\|^2_{\mathfrak{a}}\, dt + \int_0^1 \sum_{k,l=1}^{N} a_k(t)^T K(z_k(t), z_l(t)) a_l(t)\, dt + \lambda g(\theta) + \frac{1}{\sigma^2} \Gamma(F(\cdot, \theta), \mathcal{T}(1))$$

with $\partial_t z_k = \hat{v}(t, z_k(t))$, $z_k(0) = x_k$. The two approaches are, in theory, equivalent, in that the second one simply ignore the reduction on the affine part of the vector field, but it may be helpful, numerically, to use separate variables in the optimization process for the affine transform and the vector field reduced coefficients, because this gives more flexibility to the optimization. The derivation of the associated optimality conditions and gradient are similar to those made in section 3.2 and left to the reader.

Another point worth mentioning is that, if φ satisfies

$$\partial_t \varphi = g \circ \varphi + v \circ \varphi$$

for (time-dependent) $g \in \mathfrak{a}$ and $v \in V$, and if one defines the time-dependent affine transformation ρ by

$$\partial_t \rho = g \circ \rho,$$

then $\varphi = \rho \circ \psi$, where ψ satisfies $\partial_t \psi = w \circ \psi$ and $w = \rho_L^{-1} v \circ \rho$, ρ_L being the linear part of ρ. In the special case when $\mathfrak{a}$ is the Lie algebra of the Euclidean group, so that ρ is the composition of a rotation and a translation, and when the norm of V is invariant by such transformations (e.g., when using a scalar kernel associated to an RBF), then φ and ψ are equidistant to the identity. As a consequence, when the final function F implements a linear model, there is, in theory (numerics may be different), no gain in introducing an affine component restricted to rotations and translations. The equidistance property does not hold, however, if one uses a larger group of affine transformations, or a norm on V that is not Euclidean invariant, and the introduction of such transformations actually extends the model in a way that may significantly modify its performance, generally, in our experiments, for the better.

5. Enhancements and Remarks

5.1 Adding a Dimension

We use, in this paper, logistic regression as final classifier applied to transformed data. This classifier estimates a linear separation rule between the classes, but it should be clear that not every training set can be transformed into a linearly separable one using a diffeomorphism of the ambient space. A very simple one-dimensional example is when the true class associated to an input $x \in \mathbb{R}$ is 0 if $|x| < 1$ and 1 otherwise: no one-dimensional diffeomorphism will make the data separable, since such diffeomorphisms are necessarily monotone. This limitation can be fixed easily, however, by adding a dummy dimension and apply the model to a $(d+1)$-dimensional dataset in which X is replaced by $(X, 0)$. In the example just mentioned, for example, the transformation $(x, \mu) \mapsto (x, \mu + x^2 - 1)$ is a diffeomorphism of $\mathbb{R}^2$ that separates the two classes (along the y axis).

Notice that any binary classifier that can be expressed as $x \mapsto \mathrm{sign}(f(x) - a)$ for some smooth function f can be included in the model class we are considering after adding a dimension, simply taking (letting again μ denote the additional scalar variable) $\psi(x, \mu) = (x, \mu + f(x))$, which is a diffeomorphism of $\mathbb{R}^{d+1}$, and $u = (0_{\mathbb{R}^d}, 1)$. However, even when it works perfectly on the training set, this transformation will not be optimal in general, and the diffeomorphic classifier would typically prefer a diffeomorphism φ that will minimize the overall distortion, essentially trading off some non-linear transformation of the data, x, to induce a "simpler" classification rule. Figure 1 provides an example of the effect of adding a dimension when the original data is two-dimensional with two classes forming a pattern resembling a target. While no 2D transformation will break the topology and separate the central disk from the exterior ring, using an additional dimension offers a straightforward solution. Another simple example is provided in Figure 2, where a circle (class 1) is inscribed in a half ellipse (class 2). In this case, adding a dimension is not required, but the transformation estimated when this is done is closer to the identity (in the sense of our metric on diffeomorphisms) than the one computed in two dimensions.

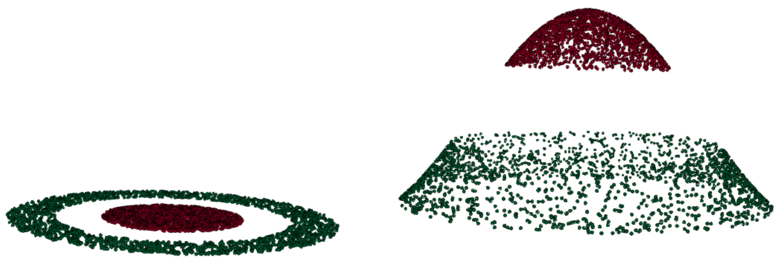

Figure 1: Additional dimension separating the center of a target from its external ring. Left: initial configuration; Right: transformed configuration.

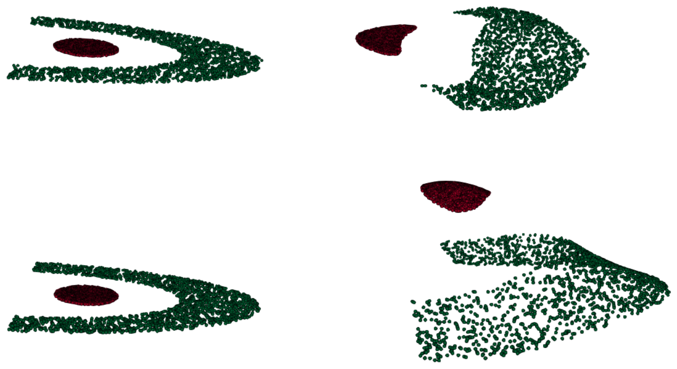

Figure 2: Comparison of optimal transformations without (top) and with (bottom) adding a dimension. Left: initial configuration; Right: transformed configuration.

When adding one or more dimensions (in a c-class problem, it makes sense to add $c-1$ dimensions), one may want to expand training data in the form $x_k \to (x_k, \delta u_k)$ for small δ, with u_k a realization of a standard Gaussian variable to help breaking symmetries in the training phase (test data being still expanded as $x_k \to (x_k, 0)$).

5.2 Parametric Dimension and Sub-Riemannian Methods

The dimensional reduction method described in section 3.1 is the exact finite-dimensional parametrization of the spatial component of the original infinite-dimensional problem. Assuming T steps for the time discretization, this results in TdN parameters in the transformation part of the model, while the computation of the gradient has a TdN^2 order cost. With our implementation (on a four-core Intel i7 laptop), we were able to handle up to 2,000 training samples in 100 dimensions with 10 time steps, which required about one day for 2,000 gradient iterations. Even if the efficiency of our implementation may be improved, for example through the use of GPU arrays, it is clear that the algorithm we just described will not scale to large datasets unless some approximations are made. Importantly, no significant reduction in computation cost can be obtained by randomly sampling from the training set at each iteration, since the representation (4) requires computing the trajectories of all sample points.

This suggests replacing the optimal expression in (4) by a form whose complexity remain bounded, rather than linear in the number of training samples. One possibility is to require that v is decomposed as a sum similar to (4), but over a smaller number of "control points", i.e., to let

$$v(t, \cdot) = \sum_{j=1}^{n} K(\cdot, \zeta_j(t)) a_j(t) \tag{18}$$

with $\partial_t \zeta_j = v(t, \zeta_j)$ and $\zeta_j(0) = \zeta_j^0$ where $\{\zeta_1^0, \ldots, \zeta_n^0\}$ is, for example, a subset of the training data set. As a result, the computational cost per iteration in the data size would now be of order $TdNn$ (because the evolution of all points is still needed to evaluate the objective function and its gradient), but this change would make possible randomized evaluations of the data cost, Γ, (leading to stochastic gradient descent) that would replace $TdnN$ by $TdnN'$ where N' is the size of the mini-batch used at each iteration. This general scheme will be explored in future work, where optimal strategies in selecting an n-dimensional subset of the training data must be analyzed, including in particular the trade-off they imply between computation time and sub-optimality of solutions. Even though they appeared in a different context, some inspiration may be obtained from similar approaches that were explored in shape analysis (Younes, 2012; Durrleman et al., 2013; Younes, 2014; Durrleman et al., 2014; Gris et al., 2018).

Another plausible choice is to specify a parametric form for the vector field at a given time. Models motivated by neural architectures have been introduced in Chen et al. (2019), and we provide in section 6.11 a few comparative results between the Riemannian approach that we introduced and a "planar" representation of the vector field (Rezende and Mohamed, 2015) in the form

$$v(t, x) = a(t)\Phi(h(t)^T x + b(t)) \tag{19}$$

where $a(t)$ and $h(t)$ are d-dimensional vectors, b is a scalar and Φ is a non-linear function (for example, $\Phi(t) = \max(t, 0)$, the positive part, or ReLU). While the exact expression of $\|v(t)\|_V$ may be challenging to compute analytically when v is given in this form, it is sufficient, in order to ensure the existence of solutions to the state equation, to control the supremum norm of the Lipschitz constant of $v(t)$. One can therefore replace $\int_0^1 \|v(t)\|_V^2 dt$ in the optimal control formulation (3) by, e.g.,

$$\int_0^1 (\alpha |a(t)|^2 + \beta |h(t)|^2 + \gamma b(t)^2) dt.$$

Note that the Lipschitz constant of $v(t, \cdot)$ is controlled (assuming that Φ is Lipschitz) by $|a(t)|\,|h(t)|$ so that, since the finiteness of the integral above ensures that $\int_0^1 |a(t)|\,|h(t)| dt$ is finite, the flow is guaranteed to provide an homeomorphic transformation. Using this restriction, the computational complexity is linear in the dimension and in the number of training samples, with significantly shorter computation times.

5.3 Deformable Templates

It is important to strengthen the fact that, even though our model involves diffeomorphic transformations, it is not a deformable template model. The latter type of model typically works with small-dimensional images (k=2 or 3), say $I : \mathbb{R}^k \to \mathbb{R}$, and tries to adjust a k-dimensional deformation (using a diffeomorphism $g : \mathbb{R}^k \to \mathbb{R}^k$) such that the deformed image, given by $I \circ g^{-1}$ aligns with a fixed template (and classification based on a finite number of templates would pick the one for which a combination of the deformation and the associated residuals is smallest).

The transformation $\psi_g : I \mapsto I \circ g^{-1}$ is a homeomorphism of the space of, say, continuous images. Once images are discretized over a grid with d points, ψ_g becomes (assuming that the grid is fine enough) a one-to-one transformation of $\mathbb{R}^d$, but a very special one. In this context, the model described in this paper would be directly applied to discrete images, looking for a d-dimensional diffeomorphism that would include deformations such as ψ_g, but many others, involving also variations in the image values or more complex transformations (including, for example, reshuffling all image pixels in arbitrary orders!).

6. Comparative Experiments

6.1 Classifiers Used for Comparison

We now provide a few experiments that illustrate some of the advantages of the proposed diffeomorphic learning method, and some of its limitations as well. We will compare the performance of this algorithm with a few off-the-shelve methods, namely k-nearest-neighbors (kNN), linear and non-linear SVM, random forests (RF), multi-layer perceptrons with 1, 2 and 5 hidden layers (abbreviated below as MLP1, MLP2 and MLP5) and logistic regression (Hastie et al., 2003; Bishop, 2006). The classification rates that were reported were evaluated on a test set containing 2,000 examples per class (except for MNIST, for which we used the test set available with this data).

We used the scikit-learn Python package (Pedregosa et al., 2011) with the following parameters (most being default in scikit-learn).

- Linear SVM: ℓ^2 penalty with default weight $C = 1$, with one-vs-all multi-class strategy when relevant.
- Kernel SVM: ℓ^2 penalty with weight C estimated as described in section 7. Gaussian (i.e., RBF) kernel with coefficient γ identical to that used for for the kernel in diffeomorphic learning. One-vs-all multi-class strategy when relevant.
- Random forests: 100 trees, with Gini entropy splitting rule, with the default choice ($\sqrt{d}$) of number of features at each node.
- k-nearest neighbors: with the standard Euclidean metric and the default (five) number of neighbors.
- Multi-layer perceptrons: with ReLU activations, ADAM solver, constant learning rate and 10,000 maximal iterations, using 1, 2 or 5 hidden layers each composed of 100 units.

- Logistic regression: ℓ^2 penalty with weight $C = 1$. The same classifier is used as the final step of the diffeomorphic learning algorithm, so that its performance on transformed data is also the performance of the algorithm that is proposed in this paper.

In all cases, we added a dummy dimension to the data as described in section 5.1 when running diffeomorphic learning (classification results on original data were obtained without the added dimension). We also used an affine transformation to complement the kernel, as described in section 4.3. Note that adding a dimension was not always necessary for learning (especially with large dimensional problems), nor was the affine transform always improving on results, but they never harmed the results in any significant way, so that it was simpler to always turn on these options in our experiments. The optimization procedure was initialized with a vanishing vector field (i.e., $\psi = \mathrm{id}$) and run until numerical stabilization (with a limit of 2,000 iterations at most). Doing so was always an overkill, in terms of classification performance, because in almost all cases, the limit classification error on the test set stabilizes faster than the time taken by the algorithm to optimize the transformation. It was however important to avoid stopping the minimization early in order to make sure that optimizing the diffeomorphism did not result in overfitting.

We now describe the datasets that we used (all but the last one being synthetic) and compare the performances of the classifiers above on the original data and on the transformed data after learning.

6.2 Tori datasets

In our first set of experiments, we let $D_i = R(\mathbb{T}_i \times \mathbb{R}^{d-3})$ where $\mathbb{T}_1$, $\mathbb{T}_2$ are non-intersecting tori in $\mathbb{R}^3$ and R is a random d-dimensional rotation. The tori are positioned as illustrated in the first panel of Figure 3, so that, even though they have an empty intersection, they are not linearly separable. The distribution of training and test data is (before rotation) the product of a uniform distribution on the torus and of a standard Gaussian in the $d-3$ remaining dimensions.

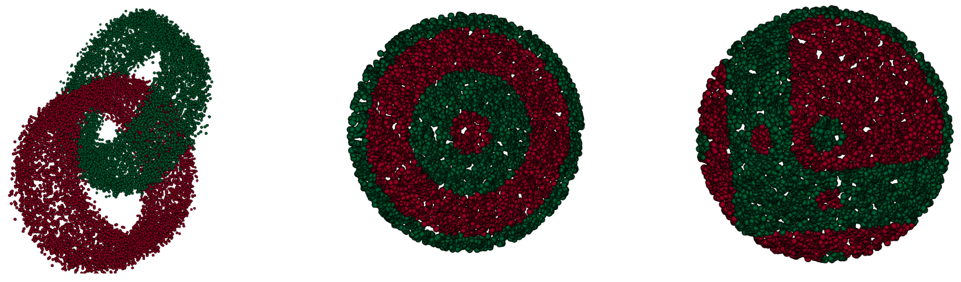

Figure 3: left: “Tori” data ($d = 3$). Center: Spherical layers ($d = 2$). Right: “RBF” data ($d = 2$).

Classification results for this dataset are summarized in Table 1. Here, we let the number of noise dimensions vary from 0 to 7 and 17 (so that the total number of dimensions is 3, 10 and 20) and the number of training samples are 200, 500 and 1,000. The problem

	Log. reg.	lin. SVM	SVM	RF	kNN	MLP1	MLP2	MLP5
	Tori, $d = 3$, 100 samples per class							
Original Data	0.341	0.341	**0.000**	0.007	**0.000**	0.004	**0.000**	**0.000**
Transformed Data	**0.000**	0.000	0.000	0.007	0.000	0.000	0.000	0.000
	Tori, $d = 10$, 100 samples per class							
Original Data	0.312	0.317	0.280	0.311	0.309	**0.159**	0.170	0.233
Transformed Data	**0.159**	0.163	0.162	0.176	0.153	0.155	0.153	0.163
	Tori, $d = 10$, 250 samples per class							
Original Data	0.325	0.326	0.207	0.285	0.257	0.073	0.093	0.109
Transformed Data	**0.023**	0.024	0.017	0.021	0.012	0.030	0.026	0.025
	Tori, $d = 20$, 100 samples per class							
Original Data	0.320	**0.317**	0.324	0.376	0.369	0.325	0.325	0.353
Transformed Data	0.321	0.329	0.322	0.347	0.330	0.320	0.319	0.323
	Tori, $d = 20$, 250 samples per class							
Original Data	0.320	0.323	0.312	0.339	0.367	**0.204**	0.284	0.280
Transformed Data	0.249	0.255	0.249	0.277	0.251	0.247	0.247	0.250
	Tori, $d = 20$, 500 samples per class							
Original Data	0.316	0.315	0.267	0.305	0.355	**0.117**	0.130	0.191
Transformed Data	0.163	0.166	0.158	0.173	0.167	0.160	0.163	0.164

Table 1: Comparative performance of classifiers on "Tori" data

becomes very challenging for most classifiers when the number of noisy dimensions is large, in which case a multi-layer perceptron with one hidden layer seems to perform best. All other classifiers see their performance improved after diffeomorphic transformation of the data. One can also notice that, after transformation, all classifiers perform approximately at the same level. Figure 4 illustrates how the data is transformed by the diffeomorphism and is typical of the other results.

We also point out that all classifiers except RF are invariant by rotation of the data, so that making a random rotation when generating it does not change their performance. The estimation of the diffeomorphism using a radial kernel is also rotation invariant. The RF classifier, which is based on comparison along coordinate axes, is highly affected, however. Without a random rotation, it performs extremely well, with an error rate of only 0.05 with 17 noisy dimensions and 250 examples per class.

6.3 Spherical Layers

We here deduce the class from the sign of $\cos(9|X|)$ where X follows a uniform distribution on the d-dimensional unit sphere, and we provide results with $d = 3$ (a representation of the data set in 2D is provided in the second panel of Figure 3). We see that linear classifiers cannot do better than chance (this is by design), but that after the estimated diffeomorphic

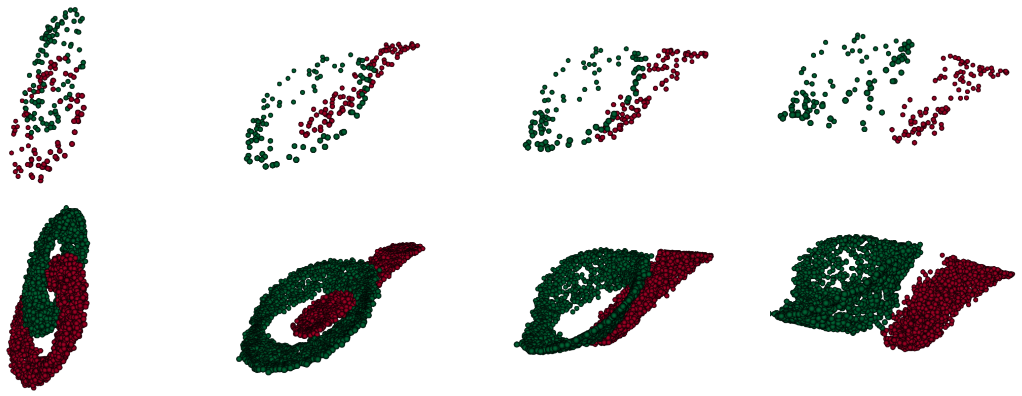

Figure 4: Visualization of the diffeomorphic flow applied to the 3D tori dataset. Left: training data; Right: test data. Top to bottom: $t = 0$, $t = 0.3$, $t = 0.7$, $t = 1$. The data is visualized in a frame formed by the discriminant direction followed by the two principal components in the perpendicular space.

transformation is applied, all classifiers outperform, with a large margin for small datasets, the best non-linear classifiers trained on the original data.

	Log. reg.	lin. SVM	SVM	RF	kNN	MLP1	MLP2	MLP5
	Spherical layers, $d = 3$, 100 samples per class							
Original Data	0.507	0.507	0.357	0.353	0.408	0.478	0.488	0.481
Transformed Data	**0.246**	0.249	0.329	0.279	0.355	0.246	0.245	0.253
	Spherical layers, $d = 3$, 250 samples per class							
Original Data	0.487	0.487	0.233	0.231	0.279	0.394	0.264	**0.113**
Transformed Data	0.119	0.118	0.185	0.129	0.199	0.149	0.143	0.138
	Spherical layers, $d = 3$, 500 samples per class							
Original Data	0.506	0.506	0.185	0.194	0.229	0.299	0.113	0.091
Transformed Data	**0.089**	0.092	0.165	0.087	0.197	0.092	0.103	0.096

Table 2: Comparative performance of classifiers on 3D spherical layers.

6.4 RBF datasets

The next dataset generates classes using sums of radial basis functions. More precisely, we let $\rho(z) = \exp(-(z/\alpha)^2)$ with $\alpha = 0.1$ and generate classes according to the sign of the function

$$\sin\left(\sum_{j=1}^{L} \rho(|X - c_j|)a_j\right) - \mu\,.$$

In this expression, X follows a uniform distribution over the d-dimensional unit sphere and μ is estimated so that both positive and negative classes are balanced. The centers, $c_1, \ldots, c_L$ are chosen as $c_j = (3j/L) e_{(j \bmod d)+1}$ where $j \bmod d$ is the remainder of the division of j by d and $e_1, \ldots, e_d$ is the canonical basis of $\mathbb{R}^d$. The coefficients are $a_j = \cos(6\pi j/L)$, and we took $L = 100$. The third panel of Figure 3 depicts the resulting regions of the unit disc in the case $d = 2$.

Table 3 provides classification performances for various combinations of dimension and sample size. Excepted when $d = 5$ and only 100 samples per class are observed, in which case linear classifiers provide the best rates, the best classification is obtained after diffeomorphic transformation.

6.5 Mixture of Gaussians

In the next example, we assume that the conditional distribution of X given $Y = y$ is normal with mean m_y and covariance matrix Σ_y. We used three classes ($y \in \{1, 2, 3\}$) in dimension 20, with

- $m_1 = (0, \ldots, 0)^T$, $m_2 = (-1, -1, -1, 0, \ldots, 0)^T$ and $m^3 = (-1, 1, -1, 0. \ldots, 0)^T$,
- $\Sigma_1 = 10\,\mathrm{Id}_{\mathbb{R}^d}$, $\Sigma_2(i,j) = 20 \exp(-|i-j|/20)$ and $\Sigma_3(i,j) = 20 \exp(-|i-j|/60)$,

where $\mathrm{Id}_{\mathbb{R}^d}$ is the d-dimensional identity matrix. Classification results for training sets with 100 and 250 examples per class are described in Table 4. While multilayer perceptrons perform best on this data, the performance of diffeomorphic classification is comparable, and improves on all other classifiers. A visualization of the transformation applied to this dataset is provided in Figure 5.

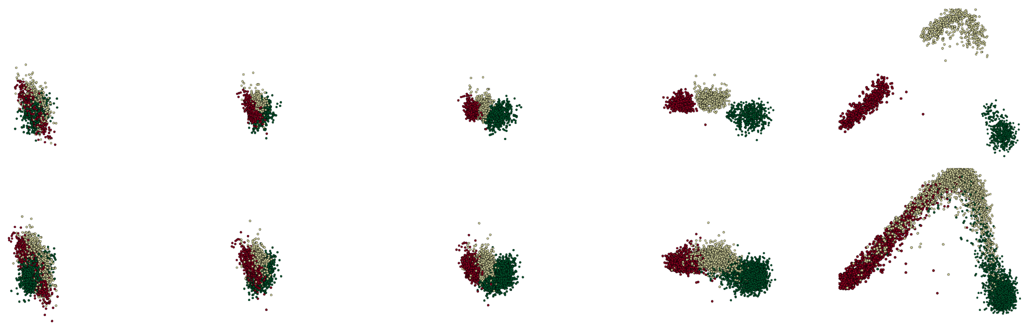

Figure 5: Diffeomorphic transformation applied to the Gaussian mixture dataset (evolution visualized at times $t = 0.0, 0.3, 0.5, 0.7$ and 1.0. The data is 20 dimensional, and the visualization is made in the plane generated by the two discriminative directions provided by logistic regression estimated on transformed data at time $t = 1$.

6.6 Curve Segments

To build this dataset, we start with two scalar functions and assume that the observed data is the restriction of one of them (where this choice determines the class) to a small discrete

	Log. reg.	lin. SVM	SVM	RF	kNN	MLP1	MLP2	MLP5
				RBF data, $d = 2$, 100 samples per class				
Original Data	0.381	0.381	0.158	**0.126**	0.208	0.183	0.193	0.163
Transformed Data	0.142	0.139	0.149	0.153	0.186	0.140	0.136	0.141
				RBF data, $d = 2$, 250 samples per class				
Original Data	0.358	0.359	0.109	0.150	0.136	0.176	0.158	0.127
Transformed Data	**0.101**	0.102	0.106	0.102	0.130	0.126	0.122	0.119
				RBF data, $d = 2$, 500 samples per class				
Original Data	0.377	0.378	0.089	0.103	0.097	0.119	0.101	0.081
Transformed Data	**0.058**	0.055	0.085	0.055	0.090	0.070	0.054	0.142
				RBF data, $d = 3$, 100 samples per class				
Original Data	0.307	0.307	0.209	0.207	0.221	0.218	0.233	0.250
Transformed Data	**0.171**	0.175	0.202	0.168	0.195	0.162	0.162	0.154
				RBF data, $d = 3$, 250 samples per class				
Original Data	0.310	0.308	0.139	0.182	0.148	0.151	0.151	0.103
Transformed Data	**0.083**	0.077	0.133	0.082	0.122	0.086	0.083	0.089
				RBF data, $d = 3$, 500 samples per class				
Original Data	0.337	0.335	0.129	0.112	0.114	0.144	0.093	0.094
Transformed Data	**0.071**	0.070	0.118	0.063	0.102	0.080	0.068	0.078
				RBF data, $d = 5$, 100 samples per class				
Original Data	**0.216**	0.218	0.221	0.256	0.254	0.280	0.259	0.233
Transformed Data	0.221	0.223	0.221	0.222	0.219	0.221	0.223	0.214
				RBF data, $d = 5$, 250 samples per class				
Original Data	0.233	0.233	**0.197**	0.226	0.213	0.250	0.273	0.240
Transformed Data	0.199	0.204	0.185	0.215	0.196	0.207	0.209	0.205
				RBF data, $d = 5$, 500 samples per class				
Original Data	0.231	0.230	0.167	0.219	0.184	0.206	0.189	0.158
Transformed Data	**0.128**	0.129	0.133	0.127	0.142	0.141	0.140	0.143

Table 3: Comparative performance of classifiers on "RBF" data

interval. More precisely, we let $Y \in \{1, 2\}$ and given $Y = y$, we let $X = (\psi_y(t-A)+\varepsilon_t, t \in I)$ where $I = \{0, 1/d, \ldots, (d-1/d)\}$ is a discretization of the unit interval, $(\varepsilon_t, t \in I)$ are independent standard Gaussian variables and A follows a uniform distribution on $[-2, 2]$. For the experiments in Table 5, we took $\psi_y(t) = \log\cosh(t/\beta_y)$ with $\beta_1 = 0.02$ and $\beta_2 = 0.021$. This table shows significant improvements of classification rates after application of the diffeomorphism.

	Log. reg.	lin. SVM	SVM	RF	kNN	MLP1	MLP2	MLP5
	Mixture of Gaussians, $d = 20$, 100 samples per class							
Original Data	0.398	0.407	0.225	0.191	0.495	0.143	0.101	**0.099**
Transformed Data	0.135	0.153	0.175	0.110	0.236	0.134	0.109	0.108
	Mixture of Gaussians, $d = 20$, 250 samples per class							
Original Data	0.354	0.359	0.172	0.163	0.427	0.073	**0.061**	0.063
Transformed Data	0.079	0.091	0.145	0.089	0.187	0.077	0.074	0.074

Table 4: Comparative performance of classifiers on Gaussian mixtures.

	Log. reg.	lin. SVM	SVM	RF	kNN	MLP1	MLP2	MLP5
	Curve segments, $d = 50$, 100 samples per class							
Original Data	0.484	0.495	0.400	0.448	0.466	0.449	0.444	0.465
Transformed Data	**0.324**	0.324	0.350	0.378	0.438	0.329	0.335	0.332
	Curve segments, $d = 50$, 250 samples per class							
Original Data	0.487	0.486	0.266	0.345	0.353	0.493	0.486	0.467
Transformed Data	**0.152**	0.157	0.147	0.195	0.248	0.155	0.258	0.216
	Curve segments, $d = 50$, 500 samples per class							
Original Data	0.495	0.496	0.155	0.294	0.230	0.505	0.484	0.426
Transformed Data	**0.093**	0.094	0.062	0.070	0.093	0.093	0.104	0.113

Table 5: Comparative performance of classifiers on "curve segment" data.

6.7 Xor

Here, we consider a 50-dimensional dataset with two classes. Each sample has all coordinates equal to zero except two of them that are equal to ± 1, and the class is 0 if the two nonzero coordinates are equal and 1 otherwise. The position of the two nonzero values is random and each takes the values ± 1 with equal probability.

Table 6 show that diffeomorphic classification performs well on this data with 100 training samples per class. When the number of examples is raised to 250 per class, multi-layer perceptrons with one or two hidden layers perform well as well. The other classifiers perform poorly even on the larger dataset. The classes are obviously not linearly separable, explaining the performances of logistic regression and linear SVMs, and the minimum distance between distinct examples from the same class is the same as that between examples from different classes, namely $\sqrt{2}$.

6.8 Segment Lengths

Our next synthetic example is a simple pattern recognition problem in $d = 100$ dimensions. Samples from class 1 take the form $(\rho_1 U_1, \ldots, \rho_d U_d)$ where ρ is uniformly distributed be-

	Log. reg.	lin. SVM	SVM	RF	kNN	MLP1	MLP2	MLP5
	xor data, $d = 50$, 100 samples per class							
Original Data	0.491	0.474	0.489	0.501	0.500	0.339	0.470	0.472
Transformed Data	**0.037**	0.005	0.039	0.007	0.008	0.012	0.009	0.014
	xor data, $d = 50$, 250 samples per class							
Original Data	0.511	0.434	0.510	0.481	0.472	0.011	0.071	0.226
Transformed Data	**0.000**	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table 6: Comparative performance of classifiers on xor data.

tween 0.75 and 1.25, and $U = (U_1, \dots U_d)$ is a binary vector with exactly l_y consecutive ones (possibly wrapping around $\{1, \dots, d\}$)and $d - l_y$ zeros, with $l_1 = 10$ and $l_2 = 11$.

An optimal linear separation between classes is achieved (because of shift invariance) by thresholding the sum of the d variables. However, because of the multiplication by $\rho_1, \dots, \rho_d$, this simple classification rule performs very poorly, while the same rule applied to the binary variables obtained by thresholding individual entries at, say, 0.5 perfectly separates the classes. It is therefore not surprising that multi-layer perceptrons perform very well on this problem and achieve the best classification rates. All other classifiers perform significantly better when run on the transformed data.

	Log. reg.	lin. SVM	SVM	RF	kNN	MLP1	MLP2	MLP5
	Segment length data, $d = 100$, 100 samples per class							
Original Data	0.462	0.412	0.387	0.491	0.505	0.337	0.343	**0.328**
Transformed Data	0.341	0.349	0.354	0.347	0.450	0.390	0.385	0.362
	Segment length data, $d = 100$, 250 samples per class							
Original Data	0.432	0.412	0.135	0.483	0.503	0.082	**0.079**	0.084
Transformed Data	0.100	0.098	0.107	0.098	0.443	0.096	0.113	0.114
	Segment length data, $d = 100$, 500 samples per class							
Original Data	0.391	0.391	0.045	0.447	0.194	**0.025**	**0.025**	**0.025**
Transformed Data	0.032	0.030	0.032	0.054	0.165	0.029	0.031	0.036

Table 7: Comparative performance of classifiers on segment lengths

6.9 Segment Pairs

This section describes a more challenging version of the former in which each data point consists in two sequences of ones in the unit circle (discretized over 50 points), these two sequences being both of length five in class 1, and of lengths 4 and 6 (in any order) in class 2. No linear rule can separate the classes and do better than chance, since such a rule would need to be based on summing the variables (by shift invariance), which returns 10 in both classes. The problem is also challenging for metric-based methods because each data point has close neighbors in the other class: the nearest non-identical neighbor in the same class is obtained by shifting one of the two segments, and would be at distance $\sqrt{2}$, which is identical to the distance of the closest element from the other class which is obtained by replacing a point in one segment by 0 and adding a 1 next to the other segment. One way to separate the classes would be to compute moving averages (convolutions) over windows of lengths 6 along the circle, and to threshold the result at 5.5, which can be represented as a one-hidden layer perceptron. As shown in Table 8, one-hidden-layer perceptrons do perform best on this data, with some margin compared to all others. Diffeomorphic classification does however improve significantly on the classification rates of all other classifiers.

	Log. reg.	lin. SVM	SVM	RF	kNN	MLP1	MLP2	MLP5
	Segment pair data, $d = 50$, 100 samples per class							
Original Data	0.477	0.466	0.476	0.475	**0.458**	0.470	0.470	0.488
Transformed Data	0.461	0.457	0.465	0.461	0.469	0.461	0.462	0.461
	Segment pair data, $d = 50$, 250 samples per class							
Original Data	0.490	0.485	0.447	0.412	0.492	**0.343**	0.392	0.405
Transformed Data	0.394	0.392	0.412	0.390	0.444	0.398	0.391	0.386
	Segment pair data, $d = 50$, 500 samples per class							
Original Data	0.501	0.502	0.369	0.320	0.504	**0.054**	0.164	0.284
Transformed Data	0.240	0.238	0.261	0.233	0.378	0.250	0.244	0.253
	Segment pair data, $d = 50$, 1000 samples per class							
Original Data	0.465	0.463	0.233	0.191	0.547	**0.008**	0.038	0.137
Transformed Data	0.071	0.068	0.099	0.073	0.281	0.085	0.082	0.085

Table 8: Comparative performance of classifiers on segment pair data

6.10 MNIST (Subset)

To conclude this section we provide (in Table 9) classification results on a subset of the MNIST digit recognition dataset (LeCun, 1998), with 10 classes and 100 examples per class for training. To reduce the computation time, we reduced the dimension of the data by transforming the original 28×28 images to 14×14, resulting in a 196-dimensional dataset. With this sample size, this reduction had little influence on the performance of classifiers run before diffeomorphic transformation. All classifiers have similar error rates, with a small improvement obtained after diffeomorphic transformation compared to linear classifiers, reaching a final performance comparable to that obtained by random forests and multi-layer perceptrons with two hidden layers. We did observe a glitch in the performance of nonlinear support vector machines, which may result from a poor choice of hyper-parameters (section 6.1) for this dataset.

	Log. reg.	lin. SVM	SVM	RF	kNN	MLP1	MLP2	MLP5
	MNIST data, $d = 196$, 100 samples per class							
Original Data	0.128	0.138	0.228	**0.111**	0.120	0.122	**0.111**	0.127
Transformed Data	0.113	0.120	0.325	0.129	0.094	0.110	0.110	0.124

Table 9: Comparative performance of classifiers on a subset of the MNIST dataset.

6.11 Comparison with neural ODEs

In Table 10, we provide classification performances obtained on the same datasets as above, when the vector fields generating diffeomorphisms are specified by (19). We still work with one added dimension, which, for neural ODEs, corresponds to the ANODE method in Dupont et al. (2019). Unsurprisingly, results vary with datasets. In some cases, ANODE achieves top performances over all classifiers: This includes tori datasets, segment lengths, and segments pairs. In some cases, such as line segments or RBF, and also MNIST, results are significantly worse that those obtained with most nonlinear classifiers.

Datasets	100	250	500	1000
3D tori	**0.00**	NA	NA	NA
10D tori	**0.15**	**0.01**	NA	NA
20D tori	**0.30**	**0.18**	**0.02**	NA
Spherical layers	0.42	0.31	0.15	NA
2D RBF	0.30	0.12	0.09	NA
3D RBF	0.11	0.06	**0.09**	NA
5D RBF	0.20	0.27	0.20	NA
M. of G.	0.13	0.11	NA	NA
Curve segments	0.48	0.49	0.32	NA
Xor	**0.035**	**0.00**	NA	NA
Segment lengths	**0.29**	**0.075**	**0.025**	NA
Segment pairs	0.46	**0.30**	**0.05**	0.015
MNIST	0.16	NA	NA	NA

Table 10: Classification rates obtained with vector fields represented using equation (19)

Note that, in order to obtain optimal results, we had to use a relatively “deep” discretization of the ODE, over 50 time steps. In spite of this, the neural net formulation is significantly faster than the “fully Riemannian” version.

6.12 Discussion

The results presented in this section are certainly encouraging, and demonstrate that the proposed algorithm has, at least, some potential. We however point out that the classifiers that were used in our experiments were run "off-the-shelves," using the setup described in section 6.1. There is no doubt that, after some tuning up specific to each dataset, each of these classifiers could perform better. Such an analysis was not our goal here, however, and the classification rates that we obtained must be analyzed with this in mind. We just note that we also used the exact same version of diffeomorphic learning for all datasets, with hyper-parameters values described in section 7.

We have included kernel-based support vector machines in our comparisons. These classifiers indeed share with the one proposed here the use of a kernel trick to reduce an infinite dimensional problem to one with a number of parameters comparable to the size of the training set. Some notable differences exist however. The most fundamental one lies in the role that is taken by the kernel in the modeling and computation. For SVMs and other kernel methods, the kernel provides, or is closely related to, the transformation of the data into a feature space. It indeed defines the inner product in that space, and can itself be considered as an infinite dimensional representation of the data, through the standard mapping $x \mapsto \kappa(x, \cdot)$. Kernel methods then implement linear methods in this feature space, such as computing a separating hyperplane for SVMs. For the diffeomorphic classification method that is described here, the kernel is used as a computational tool, similar to the way it appears in spline interpolation. The fundamental concept here is the Hilbert space V and its norm, which is, in the case of a C^k Matérn kernel, a Hilbert Sobolev space of order $k + d/2$. In small dimensions, one could actually directly discretize this norm on a finite grid (using finite differences to approximate the derivatives), as done in shape analysis for image registration (see, e.g., Beg et al. (2005)). The reproducing kernel of V is therefore a mathematical tool that makes explicit the derivation of the discrete representation described in section 3.1 rather than an essential part of the model, as it is for kernel methods such as SVMs.

Another important distinction, which makes the diffeomorphic method deviate from both kernel methods in machine learning and from spline interpolation is that the kernel (or its associated space V) is itself part of a nonlinear dynamical evolution where it intervenes at every time. This dynamical aspect of the approach constitutes a strong deviation from SVMs which applied the kernel transformation once before implementing a linear method. It is, in this regard, closer to feed-forward neural network models in that it allows for very large non-linear transformations of the data to be applied prior to linear classification. Unlike neural networks, however, diffeomorphic learning operates the transformations within the original space containing the data.

As mentioned in section 6.1, we ran the optimization algorithm until numerical convergence, although it is clear from monitoring classification over time that classes become well separated early on while the estimation of the optimal transformation typically takes more time. If one is not interested in the limit diffeomorphism (which can be of interest for modeling purposes), significant computation time may be saved by stopping the procedure earlier, using, for example a validation set.

7. Setting Hyper-Parameters

The diffeomorphic classification method described in this paper requires the determination of several hyper-parameters that we now describe with a discussion of how they were set in our experiments.

(a) Kernel scale parameter. While the kernel can be any function of positive type and therefore considered as an infinite dimensional hyper-parameter, we assume that one uses a kernel such as those described in equations (14) and (15) (our experiments use the latter with $k = 3$) and focus on the choice of the scale parameter ρ. In our implementation, we have based this selection on two constraints, namely that training data should tend to "collaborate" with some of their neighbors from the same class while data from different classes should "ignore" each other. This resulted in the following computation.

   (i) To address the first constraint, we evaluate, for each data point, the fifth percentile of its distance to other points from the same class. We then define ρ_1 to be the 75th percentile of these values.

   (ii) For the second constraint, we define ρ_2 as the 10th percentile of the minimum distance between each data point and its closest neighbor in an other class.

   We finally set $\rho = \rho_0 = \min(\rho_1, \rho_2)$. Table 11 provides a comparison of the performances of the algorithm for values of ρ that deviate from this default value, showing that, in most cases, this performance obtained with ρ_0 is not too far from the best one. The only exception was found with the Xor dataset, for which a significant improvement was obtained using a large value of ρ. Note that in Tables 11 and 12, all experiments in the same row were run with the same training and test sets.

Dataset	Samples/class	$0.25\rho_0$	$0.5\rho_0$	$0.75\rho_0$	ρ_0	$1.5\rho_0$	$2\rho_0$	$4\rho_0$
Sph. Layers	250	0.238	0.199	0.188	0.175	0.165	0.183	0.212
RBF	100	0.170	0.107	0.085	0.083	0.070	0.063	0.063
Tori ($d = 10$)	100	0.285	0.233	0.207	0.165	0.163	0.175	0.194
M. of G.	100	0.322	0.221	0.156	0.135	0.109	0.103	0.124
Xor	100	0.476	0.458	0.039	0.036	0.042	0.022	0.468
Seg. Length	250	0.164	0.143	0.114	0.097	0.093	0.093	0.86

Table 11: Performance comparison when the kernel scale deviates from the default value ρ_0, assessed on the spherical layers, RBF, Tori, mixture of Gaussians, Xor and segment length datasets.

(b) Regularization weight. A second important parameter is the regularization weight, σ^2 in Equation 1. In our experiments, this parameter is adjusted online during the minimization algorithm, starting with a large value, and slowly decreasing it until the training error reaches a target, δ. While this just replaces a parameter by another, this target value is easier to interpret, and we used $\delta = 0.005$ in all our experiments (resulting de facto in a vanishing training error at the end of the procedure in all cases).

(c) ℓ^2 penalty on logistic regression. This parameter, denoted λ in Equation 1, was taken equal to 1 in all our experiments.

(d) Time discretization. The value of T in equation (11) may also impact the performance of the algorithm, at least for small values, since one expects the model to converge to its continuous limit for large T. This expectation is confirmed in Table 12, which shows that the classification error rates obtained with the value chosen in our experiments, $T = 10$, was not far from the asymptotic one. In some cases, the error rates obtained for smaller value of T may be slightly better, but the difference is marginal.

Dataset	Samples/class	$T=1$	$T=2$	$T=4$	$T=10$	$T=20$	$T=40$	$T=100$
Sph. Layers	250	0.199	0.199	0.190	0.197	0.197	0.197	0.197
RBF	250	0.089	0.089	0.085	0.083	0.083	0.083	0.083
Tori ($d = 10$)	100	0.243	0.160	0.170	0.177	0.186	0.195	0.197
M. of G.	100	0.178	0.169	0.141	0.119	0.114	0.111	0.109
Xor	100	0.473	0.384	0.061	0.036	0.035	0.034	0.034
Seg. Length	250	0.140	0.117	0.100	0.094	0.092	0.091	0.091

Table 12: Performance comparison for various values of the number of discretization steps, T, assessed on the spherical layers, RBF, Tori, mixture of Gaussians, Xor and segment length datasets.

8. Conclusion

In this paper, we have introduced the concept of diffeomorphic learning and provided a few illustrations of its performance on simple, but often challenging, classification problems. On this class of problems, the proposed approach appeared quite competitive among other classifiers used as comparison. Some limitations also appeared, regarding, in particular, the scalability of the method, for which we have provided some options that will be explored in the future.

We have only considered, in this paper, applications of diffeomorphic learning to classification problems. Extensions to other contexts will be considered in the future. Some, such as regression, may be relatively straightforward, while others, such as clustering, or dimension reduction should require additional thinking, as the obtained results will be highly dependent on the amount of metric distortion allowed in the diffeomorphic transformation.

References

N. Aronszajn. Theory of reproducing kernels. *Trans. Am. Math. Soc.*, 68:337–404, 1950.

M. F. Beg, M. I. Miller, A. Trouvé, and L. Younes. Computing large deformation metric mappings via geodesic flows of diffeomorphisms. *Int J. Comp. Vis.*, 61(2):139–157, 2005.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Martin D. Buhmann. *Radial basis functions: theory and implementations, volume 12 of Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2003.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. *arXiv:1806.07366 [cs, stat]*, December 2019. URL `http://arxiv.org/abs/1806.07366`. arXiv: 1806.07366.

Jean Duchon. Interpolation des fonctions de deux variables suivant le principe de la flexion des plaques minces. *R.A.I.R.O. Analyse Numerique*, 10:5–12, 1977.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented Neural ODEs. In H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 3140–3150. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/8577-augmented-neural-odes.pdf`.

Stanley Durrleman, Stéphanie Allassonnière, and Sarang Joshi. Sparse adaptive parameterization of variability in image ensembles. *International Journal of Computer Vision*, 101(1):161–183, 2013.

Stanley Durrleman, Marcel Prastawa, Nicolas Charon, Julie R Korenberg, Sarang Joshi, Guido Gerig, and Alain Trouvé. Morphometry of anatomical shape complexes with dense deformations and sparse parameters. *NeuroImage*, 101:35–49, 2014.

Nira Dyn. Interpolation and approximation by radial and related functions. In C. K. Chui, L. L. Shumaker, and J. D. Ward, editors, *Approximation Theory VI: vol. 1*, pages 211–234. Academic Press, 1989.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Barbara Gris, Stanley Durrleman, and Alain Trouvé. A sub-riemannian modular framework for diffeomorphism-based analysis of shape ensembles. *SIAM Journal on Imaging Sciences*, 11(1):802–833, 2018.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Springer, 2003.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings CVPR'2016*, pages 770–778, 2016. URL `https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html`.

Leslie M Hocking. *Optimal control: an introduction to the theory with applications*. Oxford University Press, 1991.

Sarang Joshi and Michael I Miller. Landmark matching via large deformation diffeomorphisms. *IEEE transactions in Image Processing*, 9(8):1357–1370, 2000.

Ivan Kobyzev, Simon Prince, and Marcus Brubaker. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. ISSN 1939-3539. doi: 10.1109/TPAMI.2020.2992934. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

Yann LeCun. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Jack Macki and Aaron Strauss. *Introduction to optimal control theory*. Springer Science &amp; Business Media, 2012.

Jean Meinguet. Multivariate interpolation at arbitrary points made simple. *J. Applied Math. and Physics*, 30:292–304, 1979.

Mario Micheli and Joan A. Glaunès. Matrix-valued kernels for shape deformation analysis. *Geom. Imaging Comput.*, 1(1):57–139, 2014. ISSN 2328-8876; 2328-8884/e.

Michael I Miller, Alain Trouvé, and Laurent Younes. Hamiltonian systems and optimal control in computational anatomy: 100 years since d'arcy thompson. *Annual review of biomedical engineering*, 17:447–509, 2015.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 1999.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *Proceedings ICML'15*, May 2015. URL `https://arxiv.org/abs/1505.05770v6`.

François Rousseau, Lucas Drumetz, and Ronan Fablet. Residual networks as flows of diffeomorphisms. *Journal of Mathematical Imaging and Vision*, pages 1–11, 2019. Publisher: Springer.

Hadi Salman, Payman Yadollahpour, Tom Fletcher, and Kayhan Batmanghelich. Deep Diffeomorphic Normalizing Flows. *arXiv:1810.03256 [cs, stat]*, November 2018. URL `http://arxiv.org/abs/1810.03256`. arXiv: 1810.03256.

Bernhard Schölkopf and Alexander J Smola. *Learning with kernels*. MIT Press, 2002.

Alain Trouvé and Yong Yu. Metric similarities learning through examples: an application to shape retrieval. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 50–62. Springer, 2001.

Vladimir Vapnik. *The nature of statistical learning theory.* Springer science & business media, 2013.

François-Xavier Vialard, Roland Kwitt, Susan Wei, and Marc Niethammer. A Shooting Formulation of Deep Learning. *arXiv:2006.10330 [cs, math]*, June 2020. URL `http://arxiv.org/abs/2006.10330`. arXiv: 2006.10330.

Thomas L. Vincent and Walter J. Grantham. *Nonlinear and Optimal Control Systems.* Wiley, 1997.

Grace Wahba. *Spline Models for Observational Data.* SIAM, 1990.

Christian Walder and Bernhard Schölkopf. Diffeomorphic dimensionality reduction. In *Advances in Neural Information Processing Systems*, pages 1713–1720, 2009.

Laurent Younes. *Shapes and diffeomorphisms*, volume 171. Springer Science & Business Media, 2010.

Laurent Younes. Constrained diffeomorphic shape evolution. *Foundations of Computational Mathematics*, 12(3):295–325, 2012.

Laurent Younes. Gaussian diffeons for surface and image matching within a lagrangian framework. *Geometry, Imaging and Computing*, 1(1):141–171, 2014.