Diffeomorphic Learning Based on ODE Flows

Scientific work within the Ingenieurspraxis
from the Department of Electrical and Computer Engineering at the Technical University of Munich.

Supervised by	Univ.-Prof. Dr.-Ing. Sandra Hirche M. Sc. Petar Bevanda Chair of Information-Oriented Control
Submitted by	cand. Ing. Adrian Brünger Willi-Graf Str. 17/0704 80805 Munich 0049 1631925390
Submitted on	Munich, 11.04.2021

TECHNISCHE UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR INFORMATIONSTECHNISCHE REGELUNG
ORDINARIA: UNIV.-PROF. DR.-ING. SANDRA HIRCHE

April 7, 2022

I N G E N I E U R P R A X I S
for
Adrian Brünger
Student ID 03715546, Degree EI

Diffeomorphic learning based on ODE flows

Problem description:

When considering stable systems, stability guarantees are of great importance to allow for safety and physical consistency even for unseen states. Conventional learning-based approaches via function approximators such as neural networks or kernel methods for modeling dynamical systems need not reflect the system's physical properties such a stability [2]. Often, however stability conditions are enforced only locally on data points and veryfing stability in the whole state space leads to the curse of dimensionality. However, simplifying the conditions by structure and equivalences has shown to be a promising direction in recent years. For that, we use a smooth invertible transform (diffeomorphism) to fit the data to a latent linear system whose stability is trivially ensured. To learn such transforms, a method based on ordinary differential equations (ODE) solution curves is considered as it ensures invertibility. Together with a sufficiently smooth ODE learner, it forms a diffeomorphism. A seminal method employing such an approach is Neural ODE (NODE) [1], learning a neural network that delivers an invertible map when integrated over time. To simplify the training process, however, employing a kernel-based linear regressor in an inducing-state kernel approach is studied along with an approach based on random Fourier features approximating the kernel of interest. The performance of the aforementioned methods is evaluated on a handwriting dataset.

Work schedule:

- Literature research
- Designing a kernel-based learner
- Verification on a handwriting dataset
- Comparison to state-of-the-art

[1] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31, 2018.
[2] Muhammad Asif Rana, Anqi Li, Dieter Fox, Byron Boots, Fabio Ramos, and Nathan D. Ratliff. Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems. In *L4DC*, volume 120 of *Proceedings of Machine Learning Research*, pages 630–639. PMLR, 2020.

Supervisor:	M. Sc. Petar Bevanda
Start:	22.11.2021
Delivery:	11.04.2022

(S. Hirche)
Univ.-Professor

Abstract

Considering an imitation learning problem, smooth, bijective maps (diffeomorphisms) can be utilized to guarantee stability for the learned dynamic system, by linking it to a stable *latent* system. In this work, an *inducing state* kernel method and a *random Fourier feature* kernel approximation method, constructing such mappings, are presented. The kernel-based approaches are compared to the state-of-the-art *neural ordinary differential equations* method. Evaluating the approaches on a handwriting dataset, results show that the methods are capable of learning dynamic systems reproducing serverly curved trajectories, closely rembling the demonstraions. Further, due to its simplicity, the *random Fourier feature* method proves to be computationally faster then the state-of-the-art approach.

Contents

Chapter 1

Introduction

Diffeomorphic learning is an imitation learning approach, in which a stable dynamic system is learned in order to reproduce human expert demonstrations. In many robot applications, imitation learning is a natural approach for teaching a robot motions, without having to model explicit functional relationships or cost functions. Hereby, it is of major importance to guarantee stability for the learned system, to avoid uncontrolled behavior, which is especially crucial in environments involving human-robot interaction.
By introducing a smooth bijective map, a diffeomorphism, linking the target system to a stable *latent* system, one can ensure stability of the target system. When defining such a map via flows generated by ordinary differential equations (ODEs), computations of the inverse mapping are trivial. In particular when dealing with highly dynamic surroundings, fast evaluations of learned dynamics are necessary and adaptability - in terms of e.g. obstacle avoidance - is requested.

Previous approaches, such as [RLF+20, UGTP20], construct a diffeomorphism a layered fashion by composing invertible coupling neural networks. These approaches succeed in learning stable dynamic systems, however, the layered structure gives little insight in how the networks construct the diffeomorphism, which increases the difficulty of adapting the target system to changes in the environment. Another downside is computational complexity. A different approach is presented in [PSC16], which realizes a fast evaluation of the learned dynamics, while only being able to learn from one example trajectory.
This work introduces a diffeomorphic learning framework and focuses on explaining and evaluating kernel-approximation-based methods, to simplify training and minimize prediction times. Utilizing the presented framework, adaptability can be achieved by simply composing learned and customizable diffeomorphic mappings to incorporate desired behavior while maintaining stability.

1.1 Problem Setup

1.1.1 Learning Dynamical Systems from Demonstrations

Let $\mathbb{R}^d$ be a d dimensional state space. A nonlinear time-invariant first order dynamical system of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t)),\ \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{1.1}$$

can be interpreted as a mapping of a system state $\boldsymbol{x}$, e.g. cartesian position or angular coordinates of a robot manipulator, to a velocity $\dot{\boldsymbol{x}}$ through the non-linear map $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^d$. The initial state of the system is $\boldsymbol{x}_0$. Without loss of generality t_0 is set to $t_0 = 0$ in the following.
Given a set of N discretized trajectories (demonstrations) $\Xi = \{\boldsymbol{x}_{i,1}, \boldsymbol{x}_{i,2}, ..., \boldsymbol{x}_{i,l_i}\}_{i=1}^{N}$ of length l_i, with $\boldsymbol{x}_{i,j} \in \mathbb{R}^d\ \forall\, i, j$, the goal is to find a stable dynamic system (1.1), that reproduces the demonstrations best. $\boldsymbol{f}$ is constructed to be globally Lipschitz continuous to guarantee existence and uniqueness of solutions.

Assuming the velocities $\dot{\boldsymbol{x}}_{i,j}$ and $\boldsymbol{W}$ parametizes $\boldsymbol{f}$, the least squares loss

$$MSE(\boldsymbol{W}) = \sum_{i=1}^{N}\sum_{j=1}^{l_i} \|\dot{\boldsymbol{x}}_{i,j} - \hat{\dot{\boldsymbol{x}}}_{i,j}\| = \sum_{i=1}^{N}\sum_{j=1}^{l_i} \|\dot{\boldsymbol{x}}_{i,j} - \boldsymbol{f}_{\boldsymbol{W}}(\boldsymbol{x}_{i,j})\|_2^2 \tag{1.2}$$

is minimized with respect to $\boldsymbol{W}$, under the constraint that

$$\hat{\dot{\boldsymbol{x}}} = \boldsymbol{f}_{\boldsymbol{W}}(\boldsymbol{x}) \tag{1.3}$$

is stable.
Trajectory predictions are then generated by integrating the learned dynamic system (1.3).

$$\widehat{\boldsymbol{\xi}}(t, \boldsymbol{x}_0) = \boldsymbol{x}_0 + \int_0^t \hat{\dot{\boldsymbol{x}}}(u)du\ . \tag{1.4}$$

1.1.2 Stability

An eqiulibrium $\boldsymbol{x}^*$ is a state in which the velocity is zero $\dot{\boldsymbol{x}}^* = \boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{0}$. It further is a Lyapunov stable equilibrium if there exists a $\delta > 0$ for every $\epsilon > 0$, so that for any initial state $\boldsymbol{x}_0$, that satisfies $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| < \delta$, the state satisfies $\|\boldsymbol{x}(t) - \boldsymbol{x}^*\| < \epsilon$ for all $t > 0$. In other words, an equilibrium is stable if for every neighborhood U of $\boldsymbol{x}^*$ all trajectories starting in a neighborhood $V \subseteq U$ of $\boldsymbol{x}^*$ remain in U. An equilibrium is asymptotically stable if it is Lyapunov stable and all trajectories starting in V converge to the equilibrium $\lim_{t\to\infty} \|\boldsymbol{x}(t) - \boldsymbol{x}^*\| \to 0$. A dynamic system is asymptotically stable if the equilibrium is unique and $U = \mathbb{R}^d$.
Linear dynamic system of the form $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x},\ \boldsymbol{x}(0) = \boldsymbol{x}_0$, with $\boldsymbol{A}$ being non-singular, is stable for negative definite $\boldsymbol{A}$.

Chapter 2

Technical Approach

The overall goal is to learn a stable dynamic system that imitates certain motions by minimizing a distance metric between demonstrated and predicted trajectories. For $\boldsymbol{z} \in \mathbb{R}^d$ consider a gradient system

$$\dot{\boldsymbol{z}} = \boldsymbol{g}(\boldsymbol{z}) = -\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z}) \tag{2.1}$$

with convex potential

$$\Phi = \|\boldsymbol{z} - \boldsymbol{z}_{goal}\|_2^2 \,. \tag{2.2}$$

The goal of the following section is now to briefly explain the elegant approach of assuming that there exists a bijective mapping $\boldsymbol{d}$ between d-dimensional Euclidean space $\mathbb{R}^d$ (*latent* space) and a d-dimensional non-Euclidean topological space $\mathcal{M}$, so that trajectories generated by a simple, *latent* gradient descent system (2.1) in $\mathbb{R}^d$ and mapped to M closely match the demonstrated trajectories. Hereby it is presumed, that the demonstrations are generated by a globally asymptotically stable first order system with one unique equilibrium, so that is exactly linearizable by such a coordinate change $\boldsymbol{d}$.

2.1 Theoretical Foundations

2.1.1 Diffeomorphisms and Riemannian Geometry

In this section, based on [Lee06], basic concepts of Riemannian geometry and properties of diffeomorphisms are discussed to gain some intuition on the methodology used in this work. A d-dimensional manifold $\mathcal{M}$ in general is a topological space that locally resembles Euclidean space $\mathbb{R}^d$. Riemannian geometry studies Riemannian manifolds, which are smooth manifolds with a Riemannian metric. A Riemannian metric g is a positive definite 2-tensor field that determines an inner product on each tangent space of $T_p\mathcal{M}$. This is often written as $g(\boldsymbol{X}, \boldsymbol{Y}) = \langle \boldsymbol{X}, \boldsymbol{Y} \rangle$ for $\boldsymbol{X}, \boldsymbol{Y} \in T_p\mathcal{M}$. The Riemannian metric allows for definitions of geometric concepts,

such as lengths of curves and consequently distances between points on $\mathcal{M}$. A diffeomorphism $\boldsymbol{d} : \mathcal{M}_1 \to \mathcal{M}_2$ is a smoothly differentiable, bijective mapping between two smooth d-dimensional manifolds $\mathcal{M}_1$ and $\mathcal{M}_2$ with a smoothly differentiable inverse mapping $\boldsymbol{d}^{-1} : \mathcal{M}_2 \to \mathcal{M}_1$. To avoid confusion, these notations hold for the remainder of this report.

$\boldsymbol{z} \in \mathbb{R}^d$	Point/State in d-dimensional *latent* Euclidean space
$\boldsymbol{x} \in \mathcal{M}$	Point/State on d-dimensional Riemannian manifold
$\boldsymbol{d} : \mathcal{M} \to \mathbb{R}^d : \boldsymbol{d}(\boldsymbol{x}) = \boldsymbol{z}$	Diffeomorphism
$\boldsymbol{d}^{-1} : \mathbb{R}^d \to \mathcal{M} : \boldsymbol{d}^{-1}(\boldsymbol{z}) = \boldsymbol{x}$	Inverse diffeomorphism
$g : T_p\mathcal{M} \times T_p\mathcal{M} \to \mathbb{R}$	Positive definite Riemannian metric
$\boldsymbol{G} : \mathcal{M} \to \mathbb{R}^{d \times d}$	Positive definite Riemannian metric matrix
$\boldsymbol{W}$	Learning parameters

Following the approach sketched above, it can be concretized, that the diffeomorphism maps from $\mathcal{M}_1 = \mathbb{R}^d$ to $\mathcal{M}_2 = \mathcal{M}$. A Riemannian metric in euclidean space evaluated in $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathbb{R}^d$ can be represented as an inner product between the Euclidean tangent space vectors in $\boldsymbol{z}_1, \boldsymbol{z}_2$, which are the velocities $\dot{\boldsymbol{z}}_1, \dot{\boldsymbol{z}}_2$.

$$g^{\mathbb{R}}(\dot{\boldsymbol{z}}_1, \dot{\boldsymbol{z}}_2) := \dot{\boldsymbol{z}}_1^T \dot{\boldsymbol{z}}_2 =:= \dot{\boldsymbol{z}}_1^T \boldsymbol{I} \dot{\boldsymbol{z}}_2 \tag{2.3}$$

$\boldsymbol{I} \in \mathbb{R}^{d \times d}$ is the identity matrix. Since the diffeomorphism $\boldsymbol{d}(\boldsymbol{x}) = \boldsymbol{z}$ is differentiable, a tangent space vector/velocity vector on the manifold can be written as

$$\frac{d}{dt}\boldsymbol{d}(\boldsymbol{x}(t)) = \boldsymbol{J}_{\boldsymbol{d}}(\boldsymbol{x}(t))\dot{\boldsymbol{x}}(t) = \dot{\boldsymbol{z}}(t) \; . \tag{2.4}$$

With $\boldsymbol{J}_{\boldsymbol{d}}(\boldsymbol{x}(t))$ being the Jacobian of the diffeomorphism evaluated in $\boldsymbol{x}(t)$. In the following time dependencies and state dependencies of Jacobians are hidden to simplify notation. Utilizing (2.4) a Riemannian metric $g^{\mathcal{M}}$ of the Riemannian manifold $\mathcal{M}$ can be defined as

$$\begin{aligned} g^{\mathbb{R}}(\dot{\boldsymbol{z}}_1, \dot{\boldsymbol{z}}_2) &= g^{\mathbb{R}}\left(\frac{d}{dt}\boldsymbol{d}(\boldsymbol{x})|_{\boldsymbol{x}_1}, \frac{d}{dt}\boldsymbol{d}(\boldsymbol{x})|_{\boldsymbol{x}_2}\right) = \left(\frac{d}{dt}\boldsymbol{d}(\boldsymbol{x})|_{\boldsymbol{x}_1}\right)^T \frac{d}{dt}\boldsymbol{d}(\boldsymbol{x})|_{\boldsymbol{x}_2} \\ &= (J_{\boldsymbol{d}}\dot{\boldsymbol{x}}_1)^T J_{\boldsymbol{d}}\dot{\boldsymbol{x}}_2 = \dot{\boldsymbol{x}}_1^T J_{\boldsymbol{d}}^T J_{\boldsymbol{d}}\dot{\boldsymbol{x}}_2 = \dot{\boldsymbol{x}}_1^T \boldsymbol{G}\dot{\boldsymbol{x}}_2 \\ &:= g^{\mathcal{M}}(\dot{\boldsymbol{x}}_1, \dot{\boldsymbol{x}}_2) \; . \end{aligned} \tag{2.5}$$

Here, G is the positive definite Riemannian metric matrix associated with $g^{\mathcal{M}}$, which - in this case ($g^{\mathbb{R}}, \boldsymbol{d} \in \mathbb{C}^1$) - also is the *pullback* of $g^{\mathbb{R}}$ by $\boldsymbol{d}$. In differential geometry there are *pushforwards* and *pullbacks*. Using a map, in this case the diffeomorphism $\boldsymbol{d} : \mathcal{M} \to \mathbb{R}^d$, with $\mathcal{M} \to \mathbb{R}^d$ being the "forward direction", it can be observed, that points are "pushed forward" by $\boldsymbol{d}$ (as well as velocities by $\dot{\boldsymbol{d}}$) while functions are "pulled back", $g^{\mathbb{R}} : \mathbb{R} \to \mathbb{R}$ and $g^{\mathbb{R}} \circ \boldsymbol{d} : M \to \mathbb{R}$. In other words $g^{\mathcal{M}}$ is the metric on the manifold M induced by the map $\boldsymbol{d}$, by "pulling" the metric $g^{\mathbb{R}}$ from $\mathbb{R}$ onto

$\mathcal{M}$ through $\boldsymbol{d}$.
An important observation is that $\boldsymbol{d}$ is an isometry due to $g^{\mathbb{R}} \circ \boldsymbol{d} = g^{\mathcal{M}}$. This distance preserving feature of $\boldsymbol{d}$ can be sketched as follows. On a Riemannian manifold $\mathcal{M}$ the *distance* between to points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{M}$ can be defined as the infimum length of all piecewise continuous paths $\boldsymbol{\gamma}^{\mathcal{M}} : [t_1, t_2] \to \mathcal{M}$ connecting $\boldsymbol{x}_1, \boldsymbol{x}_2$:

$$d(\boldsymbol{x}_1, \boldsymbol{x}_2) = \inf\{L(\boldsymbol{\gamma}^{\mathcal{M}}) \mid \boldsymbol{\gamma}^{\mathcal{M}} : [t_1, t_2] \to \mathcal{M},\ \boldsymbol{\gamma}^{\mathcal{M}}(t_1) = \boldsymbol{x}_1,\ \boldsymbol{\gamma}^{\mathcal{M}}(t_2) = \boldsymbol{x}_2\} \tag{2.6}$$

L is a cost functional assigning a length scalar to each input path:

$$\begin{aligned} L(\boldsymbol{\gamma}^{\mathcal{M}}) &= \int_{t_1}^{t_2} \sqrt{g^{\mathcal{M}}(\dot{\boldsymbol{\gamma}^{\mathcal{M}}}(t), \dot{\boldsymbol{\gamma}^{\mathcal{M}}}(t))} dt \\ &= \int_{t_1}^{t_2} \sqrt{g^{\mathbb{R}}(\dot{\boldsymbol{\gamma}^{\mathbb{R}}}(t), \dot{\boldsymbol{\gamma}^{\mathbb{R}}}(t))} dt \\ &= L(\boldsymbol{\gamma}^{\mathbb{R}}) \\ &= L(\boldsymbol{d}(\boldsymbol{\gamma}^{\mathcal{M}})) \end{aligned} \tag{2.7}$$

So the length of each path $\boldsymbol{\gamma}^{\mathcal{M}}$ on $\mathcal{M}$ between $\boldsymbol{x}_1, \boldsymbol{x}_2$ equals the length of its mapping to Euclidean space $\boldsymbol{\gamma}^{\mathbb{R}}$ between $\boldsymbol{z}_1, \boldsymbol{z}_2$ via $\boldsymbol{d}^{-1}$. The equation (2.7) holds since (2.5) holds and $\boldsymbol{d}^{-1}$ is continuously differentiable, which allows to invert (2.4):

$$\frac{d}{dt}\boldsymbol{d}^{-1}(\boldsymbol{z}) = \boldsymbol{J}_{\boldsymbol{d}}^{-1}\dot{\boldsymbol{z}} = \dot{\boldsymbol{x}}\ . \tag{2.8}$$

It leads to the conclusion that $\boldsymbol{d}$ maps L-minimizing paths (shortest paths/geodesics) $\boldsymbol{\gamma}^{\mathcal{M}}$ on between $\boldsymbol{x}_1, \boldsymbol{x}_2$ to L-minimizing paths (shortest paths/geodesics) $\boldsymbol{\gamma}^{\mathbb{R}}$ between $\boldsymbol{z}_1, \boldsymbol{z}_2$. This intuition is further exploited in [2.1.2]. Furthermore, mapped points have the same distance in both spaces

$$\begin{aligned} l(\boldsymbol{x}_1, \boldsymbol{x}_2) &= \inf\{L(\boldsymbol{\gamma}^{\mathbb{R}}) \mid \boldsymbol{\gamma}^{\mathbb{R}} : [t_1, t_2] \to \mathbb{R}^d,\ \boldsymbol{\gamma}^{\mathbb{R}}(t_1) = \boldsymbol{z}_1,\ \boldsymbol{\gamma}^{\mathbb{R}}(t_2) = \boldsymbol{z}_2\} \\ &= l(\boldsymbol{z}_1, \boldsymbol{z}_2)\ . \end{aligned} \tag{2.9}$$

Another extremely useful feature of diffeomorphisms is that compositions of diffeomorphisms are diffeomorphisms, which allows for a layered construction of diffeomorphisms as in [RLF+20].

2.1.2 Natural Gradient Descent

A simple approach to generating *paths* of a desired shape would be to assume that the paths of the given trajectories are geodesics on some manifold $\mathcal{M}$ and to learn a diffeomorphism $\boldsymbol{d}_{\boldsymbol{W}}$ that maps from the manifold to Euclidean space. By mapping the start and goal point to Euclidean space one could then generate the desired path on the manifold by mapping a straight line (geodesic in $\mathbb{R}^d$), connecting the mapped start and goal point, back onto the manifold via $\boldsymbol{d}_{\boldsymbol{W}}^{-1}$ as shown in Figure

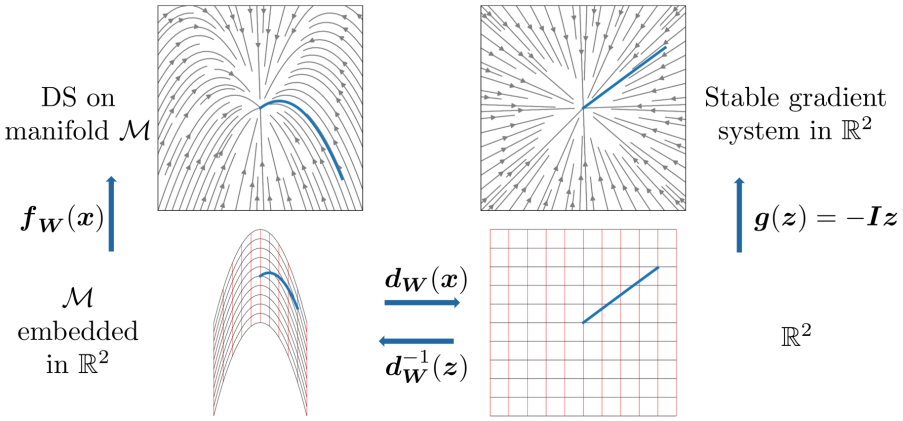

Figure 2.1: Sketch of generating a desired trajectory by a learned dynamic system (DS) on a manifold $\mathcal{M}$ (*upper left*), by mapping a trajectory, generated by a stable, *latent* gradient system in $\mathbb{R}^d$ (*upper right*) onto M via a learned diffeomorphism $\boldsymbol{d_W^{-1}}$ (*lower right* to *lower left*).

2.1. Note that a straight line in $\mathbb{R}^d$ is can be interpreted as a trajectory generated by a (stable) gradient system (2.1).

Now, this approach is extended in order to be able to learn a dynamic system that generates trajectories, whose paths are geodesics on $\mathcal{M}$ that match the example trajectories and traces them in similar time. In other words, the velocities along the given and generated trajectories should match. To achieve this, one can further note, that in a gradient system $\dot{\boldsymbol{z}} = -\nabla_{\boldsymbol{z}}\Phi(\boldsymbol{z})$ the velocity in a point $\boldsymbol{z}$ equals the steepest descent of the potential Φ in $\boldsymbol{z}$. To mimic this behavior on $\mathcal{M}$, the goal is to find a similar gradient system $\dot{\hat{\boldsymbol{x}}} = \boldsymbol{f_W}(\boldsymbol{x})$ on $\mathcal{M}$, which modifies the steepest gradient on the potential Φ of a mapped point $\boldsymbol{d_W}(\boldsymbol{x})$ in $\mathbb{R}^d$ so that it matches some given velocity $\dot{\boldsymbol{x}}$, which then is the steepest gradient on $\mathcal{M}$, respective the potential $\Phi \circ \boldsymbol{d_W}$. Such a system is called *Natural Gradient System* and can be formally written as:

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{f_W}(\boldsymbol{x}) = -\boldsymbol{G}^{-1}(\boldsymbol{x})\nabla_{\boldsymbol{x}}\Phi(\boldsymbol{d_W}(\boldsymbol{x})) = -\nabla_{\boldsymbol{G}}\Phi(\boldsymbol{d_W}(\boldsymbol{x})) \tag{2.10}$$

One can show [Ama98] that the negative *Natural Gradient* $-\nabla_{\boldsymbol{G}}\Phi(\boldsymbol{d_W}(\boldsymbol{x}))$ is the steepest descent of $\Phi \circ \boldsymbol{d_W}$ on $\mathcal{M}$.
With the potential $\Phi = \|\boldsymbol{z} - \boldsymbol{z}_{goal}\|^2_{\boldsymbol{A}}$, weighted with a positive negative $\boldsymbol{A}$, (2.10) can be written as

$$\boldsymbol{f_W}(\boldsymbol{x}) = -\boldsymbol{J}_{\boldsymbol{d_W}}\boldsymbol{A}\boldsymbol{d_W}(\boldsymbol{x}) \overset{\boldsymbol{A}=\boldsymbol{I}}{=} \boldsymbol{J}_{\boldsymbol{d_W}}\boldsymbol{I}\boldsymbol{d_W}(\boldsymbol{x}) \,, \tag{2.11}$$

due to the potential choice (2.2) in this work.

2.1.3 Stability

An intuition on why the *Natural Gradient System* is stable if the *latent* gradient system is stable can be gained as follows. Since the *latent* system is stable, all

paths of trajectories in *latent* space are straight lines starting in a mapped state $\boldsymbol{z}_0 = \boldsymbol{d}_{\boldsymbol{W}}(\boldsymbol{x}_0)$ and ending in an equilibrium $\boldsymbol{z}^* = \boldsymbol{z}_{goal}$. Since $\boldsymbol{d}$ is bijective, every mapped-back path on $\mathcal{M}$ ends in $\boldsymbol{x}^* = \boldsymbol{d}_{\boldsymbol{W}}^{-1}(\boldsymbol{z}^*)$. Furthermore, since every trajectory in *latent* space is generated by moving along the steepest gradient and converges towards $\boldsymbol{z}^*$, every trajectory on M, generated by moving along the steepest gradient, will converge towards $\boldsymbol{x}^*$. This feature is proven in [RLF+20] and guarantees stability of the learned system by structure.

2.1.4 Flow-based Construction of Diffeomorphisms

To guarantee diffeomorphic properties and a straight forward computation of the inverse mapping $\boldsymbol{d}_{\boldsymbol{W}}^{-1}$, the diffeomorphism is constructed in a flow-based fashion. Given the ordinary differential equation (ODE)

$$\dot{\boldsymbol{\gamma}}(\boldsymbol{s}, t) = \boldsymbol{V}_{\boldsymbol{W}}(\boldsymbol{\gamma}(\boldsymbol{s}, t), t) \tag{2.12}$$

a diffeomorphism can be defined as

$$\boldsymbol{d}_{\boldsymbol{W}}(\boldsymbol{x}) = \boldsymbol{\gamma}(\boldsymbol{x}, T) = \boldsymbol{x} + \int_0^T \boldsymbol{V}_{\boldsymbol{W}}(\boldsymbol{\gamma}(\boldsymbol{x}, t))dt = \boldsymbol{z} \tag{2.13}$$

if $\boldsymbol{VW}$ is a smoothly differentiable vector field from some *separable Hilberspace* $\mathcal{H}$ [DGM98]. In other words, the diffeomorphism can be defined as a smooth flow generated by integrating the vector field $\boldsymbol{V}_{\boldsymbol{W}}$ for time T. As in 1.1.1, due to smooth differentiability, every partial derivative of $\boldsymbol{V}_{\boldsymbol{W}}$ is bounded on every bounded neighborhood. Therefore $\boldsymbol{V}_{\boldsymbol{W}}$ is Lipschitz continuous. The existence and uniqueness of a solution to the ODE and therefore the diffeomorphism follows by applying Picard-Lindelöf. By noting that for any initial point $\boldsymbol{s}_0 \in \mathcal{M}$ and time t an inverse of $\boldsymbol{s}_t = \boldsymbol{d}_{\boldsymbol{W}}(\boldsymbol{s}_0) = \boldsymbol{\gamma}(\boldsymbol{s}_0, t)$ can be defined as $\boldsymbol{s}_0 = \boldsymbol{d}_{\boldsymbol{W}}^{-1}(\boldsymbol{s}_t) = \boldsymbol{\gamma}(\boldsymbol{s}_t, -t)$. That leads to the following definition of the inverse diffeomorphism:

$$\boldsymbol{d}_{\boldsymbol{W}}^{-1}(\boldsymbol{z}) = \boldsymbol{\gamma}(\boldsymbol{z}, -T) = \boldsymbol{z} + \int_{-T}^0 \boldsymbol{V}_{\boldsymbol{W}}(\boldsymbol{\gamma}(\boldsymbol{x}), t)dt = \boldsymbol{x} \tag{2.14}$$

2.2 Infinitesimal Generators

In the following, three approaches are shown to define $\boldsymbol{V}_{\boldsymbol{W}}$, utilized to implicitly learn diffeomorphisms $\boldsymbol{d}_{\boldsymbol{W}}$.

2.2.1 Kernel Methods

To learn a scalar nonlinear function, in this case $y = v_{\boldsymbol{w}}(\boldsymbol{s})$, *kernel methods* [HSS08] can be employed to implicitly use a very high dimensional feature space $\mathcal{H}$, in which the targets y may be a linear function of the mapped input $\boldsymbol{\varphi}(\boldsymbol{s})$, $\boldsymbol{\varphi} : \mathcal{M} \to \mathcal{H}$. That

allows for utilizing simple methods such as *linear regression* in the *lifted* domain $\mathcal{H}$. Considering a *linear regression* approach of the form

$$\hat{y} = v_{\widehat{\boldsymbol{w}}}(\boldsymbol{s}) = \langle \widehat{\boldsymbol{w}}, \boldsymbol{s} \rangle \ . \tag{2.15}$$

Given a data matrix $\boldsymbol{S} \in \mathbb{R}^{N \times d}$ with full column rank d (linearly independent data points $\{\boldsymbol{s}\}_{i=1}^{N} \in \mathcal{M}$ as rows), corresponding to a target vector $\boldsymbol{y}$, $\widehat{\boldsymbol{w}}$ can be obtained by minimizing the mean square error $\hat{R}$ with respect to $\boldsymbol{w}$

$$\hat{R}(\boldsymbol{w}) = \frac{1}{N} \|\boldsymbol{S}^T \boldsymbol{w} - \boldsymbol{y}\|_2^2 \tag{2.16}$$

as follows.

$$\min_{\boldsymbol{w}} (\hat{R}(\boldsymbol{w})) \implies \widehat{\boldsymbol{w}} = (\boldsymbol{S}^T \boldsymbol{S})^{-1} \boldsymbol{S}^T \boldsymbol{y} \ . \tag{2.17}$$

Which allows to make a prediction $\hat{y}$ by

$$\begin{aligned} \hat{y} &= \widehat{\boldsymbol{w}}^T \boldsymbol{s} \\ &= \left[\boldsymbol{S}^T \boldsymbol{S})^{-1} \boldsymbol{S}^T \boldsymbol{y} \right]^T \boldsymbol{s} \\ &= \boldsymbol{s}^T \boldsymbol{S}^T (\boldsymbol{S} \boldsymbol{S}^T)^{-1} \boldsymbol{y} \end{aligned} \tag{2.18}$$

Applying the feature map $\boldsymbol{\varphi}$ and noting that the prediction $\widehat{y}$ only depends on pairs of mapped input points, the *Representer Theorem* allows for an implicit formulation of the form

$$\boldsymbol{v}_{\boldsymbol{w}}(\boldsymbol{s}) = \langle \boldsymbol{w}, \boldsymbol{\varphi}(\boldsymbol{s}) \rangle_{\mathcal{H}} = \sum_{i=1}^{N} \alpha_i \boldsymbol{k}(\boldsymbol{s}, \boldsymbol{s_i}) \tag{2.19}$$

with $\boldsymbol{k} : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$, $\boldsymbol{k}(\boldsymbol{s}, \boldsymbol{s}') = \langle \boldsymbol{\varphi}(\boldsymbol{s}), \boldsymbol{\varphi}(\boldsymbol{s}') \rangle_{\mathcal{H}}$ being the positive definite *kernel* corresponding to the feature map $\boldsymbol{\varphi}$. Note, that if $\boldsymbol{\varphi}$ is infinite dimensional, $\boldsymbol{w}$ would have to be infinite dimensional as well, which is infeasible. Utilizing the *Representer Theorem* reduces the number of learnable parameters to N, depending only on the number of input data-points. This can be shown for the sketched linear regression approach by rewriting (2.18) as

$$\hat{y} = \boldsymbol{\kappa}(\boldsymbol{s}) \boldsymbol{K}(\boldsymbol{S}, \boldsymbol{S})^{-1} \boldsymbol{y} \tag{2.20}$$

with $\boldsymbol{\kappa}_i = \boldsymbol{k}(s, s_i)$ and the kernel matrix $\boldsymbol{K} : K_{i,j} = \boldsymbol{k}(s_i, s_j)$. Here, the expression $\boldsymbol{K}(\boldsymbol{S}, \boldsymbol{S})^{-1} \boldsymbol{y} \in \mathbb{R}^N$ represents the learned implicit N weights α_i from (2.19).

In 2 dimensions $\boldsymbol{y} = \boldsymbol{V}_{\boldsymbol{W}}$ can be defined element-wise as

$$\boldsymbol{V}_{\boldsymbol{W}}(\boldsymbol{s}) = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} \langle \boldsymbol{w}_1, \boldsymbol{\varphi}(\boldsymbol{s}) \rangle_{\mathcal{H}} \\ \langle \boldsymbol{w}_2, \boldsymbol{\varphi}(\boldsymbol{s}) \rangle_{\mathcal{H}} \end{bmatrix} = \langle \boldsymbol{W}, \boldsymbol{\varphi}(\boldsymbol{s}) \rangle_{\mathcal{H}} \ . \tag{2.21}$$

In the following approaches the Gaussian radial-basis function (RBF) kernel

$$\boldsymbol{k}(\boldsymbol{s}, \boldsymbol{s}') = \exp(-l \|\boldsymbol{s} - \boldsymbol{s}'\|_2^2) \tag{2.22}$$

is used due to its smoothness properties [JHS+15]. The tunable parameter $l > 0$ can be interpreted as the *kernel width*.
To avoid having to compute the inverse of the kernel matrix $\boldsymbol{K}$ as in (2.20), which has a computational complexity of $\mathcal{O}(N^3)$, two kernel-approximation approaches are presented.

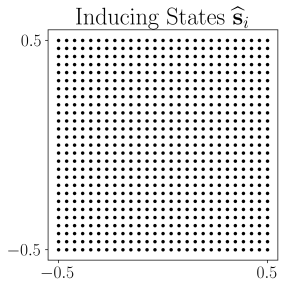

Figure 2.2: Equally spaced grid of $m = 729$ *inducing states on* $[-0.5, 0.5] \times [-0.5, 0.5]$.

Induced Gaussian Kernel (IGK)

A first, rather primitive approach to approximate the Gaussian kernel is to define the vector field as an inner product between a weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times d}$ and a *kernel vector*

$$\boldsymbol{V}_{\boldsymbol{W}(}\boldsymbol{s}) = \boldsymbol{W}^T \widetilde{\boldsymbol{k}}(\boldsymbol{s}, \widehat{\boldsymbol{s}}),\ \widetilde{\boldsymbol{k}}(\boldsymbol{s}, \widehat{\boldsymbol{s}}) = \begin{bmatrix} \exp(l\|\boldsymbol{s} - \widehat{\boldsymbol{s}}_1\|_2^2) \\ \exp(l\|\boldsymbol{s} - \widehat{\boldsymbol{s}}_2\|_2^2) \\ \dots \\ \exp(l\|\boldsymbol{s} - \widehat{\boldsymbol{s}}_m\|_2^2) \end{bmatrix} . \tag{2.23}$$

The states $\widehat{\boldsymbol{s}}$, called *inducing states*, are predefined states on an equally spaced grid, covering the range of demonstrations. An example is shown in Figure 2.2.

Random Fourier Features (RFF)

To circumvent specifying *inducing states* as in the previous section, one can use *random Fourier features* (RFF) to approximate the Gaussian kernel as shown in [RR07]

$$\boldsymbol{V}_{\boldsymbol{W}(}\boldsymbol{s}) = \boldsymbol{W}^T \boldsymbol{\tau}(\boldsymbol{s}),\ \boldsymbol{\tau}(\boldsymbol{s}) = \sqrt{\frac{2}{m}} \begin{bmatrix} \cos(\boldsymbol{\alpha}_1^T \boldsymbol{s} + \boldsymbol{\beta}_1) \\ \cos(\boldsymbol{\alpha}_2^T \boldsymbol{s} + \boldsymbol{\beta}_2) \\ \dots \\ \cos(\boldsymbol{\alpha}_m^T \boldsymbol{s} + \boldsymbol{\beta}_m) \end{bmatrix},\ \begin{matrix} \boldsymbol{\alpha}_i \sim \mathcal{N}(\boldsymbol{0}, l\boldsymbol{I}) \\ \boldsymbol{\beta}_i \sim \mathcal{U}(0, 2\pi) \end{matrix} . \tag{2.24}$$

The approximation $\langle \boldsymbol{\varphi}(\boldsymbol{s}), \boldsymbol{\varphi}(\boldsymbol{s}') \rangle_{\mathcal{H}} \approx \boldsymbol{\tau}(\boldsymbol{s})^T \boldsymbol{\tau}(\boldsymbol{s}')$, $\boldsymbol{\tau}$ being an m dimensional vector as described in (2.24), avoids computing the inverse kernel matrix by instead computing an inverse RFF matrix $(\boldsymbol{\tau}(\boldsymbol{s})^T \boldsymbol{\tau}(\boldsymbol{s}'))^{-1}$, which has a computational complexity of $\mathcal{O}(m^3)$. With $N \gg m$ a significant increase in performance can be achieved.

2.2.2 Neural Ordinary Differential Equations (NODE)

Neural Ordinary Differential Equations [CRBD18] can be seen as the state-of-the-art approach to ODE-based diffeomorphic learning. Utilizing NODE, $\boldsymbol{V}_{\boldsymbol{W}}$ is defined as a Neural Network with smooth activation functions, to guarantee that the vector field is smooth as desired.

$$\boldsymbol{V}_{\boldsymbol{W}(}\boldsymbol{s}) = \boldsymbol{NN}_{\boldsymbol{W}}^{ELU}(\boldsymbol{s}) \tag{2.25}$$

2.3 Implementation

For implementing the three methods, the Python `PyTorch` library is used. The code is CUDA optimized to run on a graphics card for accelerating matrix operations. ADAM [KB14] is chosen for optimzing and gradient computations are performed using the PyTorch `autograd` package. ODE solvers are chosen from the `torchdiffeq` library.

Chapter 3

Evaluation and Conclusion

The presented algorithms are evaluated on the LASA handwriting dataset, consisting of 30 human handwriting motions. For each motion/shape, 7 trajectories of 1000 data points are given.
Firstly, determined heuristically, the five most complex motions are chosen in order to test and compare the three methods. The selection can be seen in Figure **??** and focuses on either motions with heavily curved trajectories, such as the second, third and last shape, motions represented by frequently intersecting trajectories (fourth shape), or shapes with closely distanced starting and goal points.
Secondly, the data is preprocessed to remove noise and to smooth the trajectories as follows. The trajectories are downsampled and filtered. Further, the data points at the beginning and end of each trajectory are dropped to avoid random deflections when placing the pen on -or lifting the pen from the drawing surface. Lastly the velocities along the trajectories are recomputed via infinitesimal differences to circumvent noisy velocity sensor data. After preprocessing each trajectory consists of $l_i = 235$ points, resulting in a total of $N = 1645$ training points.

3.1 Experimental Results and Performance

To account for a *fair* training set-up, every method is trained on the $N = 1645$ data points for 5000 epochs. The kernel methods are constructed with $m = 900$ inducing states/RFFs respectively, to approximate a Gaussian RBF kernel with a *kernel width* $l = 60$. The NODE model has a layer structure of $[2, 50, 50, 2]$. For generating the diffeomorphism as in 2.1.4, 4 steps of a fourth order Runge Kutta (RK4) method are used for approximating a solution to the ODE in the time-frame $t \in [0, 2s]$.
Analyzing the test results, one can see that every approach is able to learn stable nonlinear systems, that are able to reproduce severely curved trajectories. One can further notice that the learned first order dynamical systems are incapable of modeling intersections in the data. To measure the reproduction quality, the root mean square error (RMSE) between generated and example trajectories is computed.

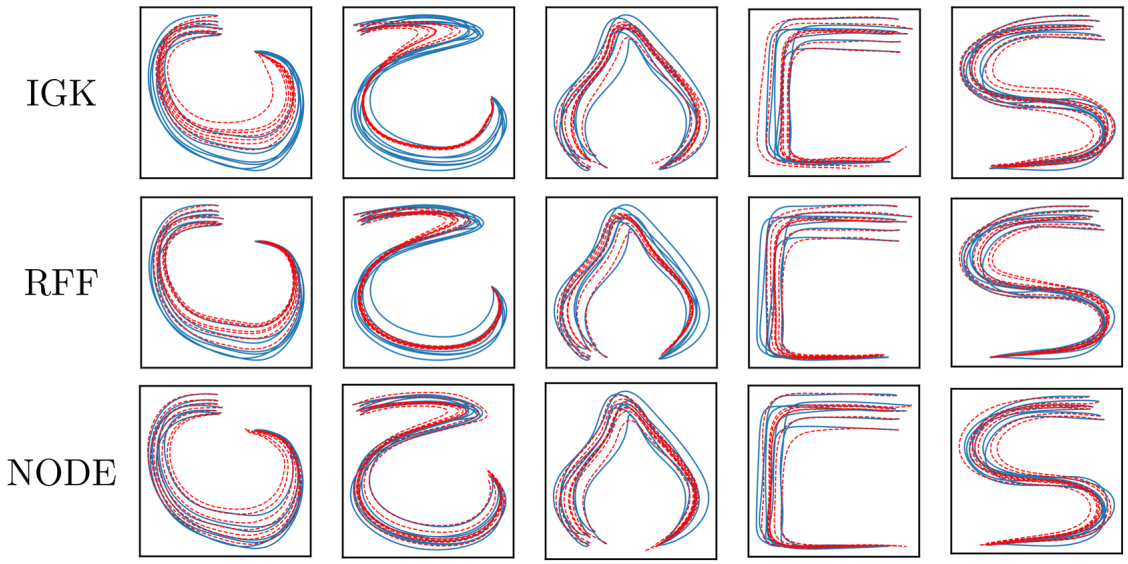

Figure 3.1: Five LASA handwriting shapes, human expert demonstrations (blue) and reproduced trajectories by learned dynamic systems (red). Induced gaussian kernel (IGK) approach is evaluated in *first row*, random Fourier feature (RFF) approach in *second row* and neural ordinary differential equation (NODE) approach in *third row*.

By examining the mean RMSE per shape in Figure 3.2, one can see that the RFF approach performs similarly well compared to the state-of-the-art NODE approach. The IGK approach has a slightly worse reproduction quality.

To compare computational performance, the respective algorithms are run on a *NVIDIA TITAN V* graphics card for 1000 epochs. The training time per epoch includes a prediction/evaluation of the learned dynamics and the computation of a gradient descent step to update the weights $\boldsymbol{W}$. In Figure 3.2 the averaged training and prediction times per epoch are shown respective the fastest approach, which is the RFF method.

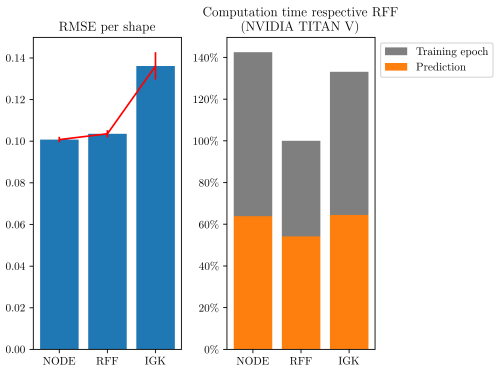

Figure 3.2: Mean RMSE per shape (*left*) and computation times respective RFF approach (*right*).

3.2 Conclusion and Future Work

Overall, regarding reproduction quality, the RFF and the state-of-the-art NODE approach perform similarily well and with some further tuning of the *kernel width* l a slight performance increase of the RFF approach is most likely possible. Respective computation time, a large margin is caused by the computation of the batch-wise inverse Jacobian $\boldsymbol{J}_{\boldsymbol{dW}}^{-1}$, which is furthermore computed multiple times in each ODE-

solving step. Therefore, when reducing the number of steps, a major decrease in computation time could be achieved. In 3.1, using a simple explicit Euler step approximating the ODE solution, the number of computations of $\boldsymbol{J}_{\boldsymbol{dW}}^{-1}$ can be reduced from 16 (4 RK4 steps) to 1. With some further testing one may find a better trade-off between approximation quality of the ODE (in terms of integration methods and number of steps) and computation time. Futhermore the computation time of the RFF approach may be improved without loosing reproduction quality by reducing the number of features (probably also by a large margin).
Structurally, the methods may be improved by further learning a positive definite $\boldsymbol{A}$ matrix (2.11), corresponding to the *latent* system $\boldsymbol{g}(\boldsymbol{z}) = -\boldsymbol{A}\boldsymbol{z}$, to reduce the *deformation* the diffeomorphism has to cope for. To be able to learn systems, that can model intersections in the data, second order dynamical systems may be considered.
Concluding, the RFF and NODE approach both perform well, with the RFF approach offering a lot of fine-tuning options and future work, which might make it the method of choice for learning stable dynamic systems, especially if fast evaluations of the dynamics are desired.

Bibliography

[Ama98] Shun-ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, 02 1998. URL: https://doi.org/10.1162/089976698300017746, arXiv:https://direct.mit.edu/neco/article-pdf/10/2/251/813415/089976698300017746.pdf, doi:10.1162/089976698300017746.

[CRBD18] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *CoRR*, abs/1806.07366, 2018. URL: http://arxiv.org/abs/1806.07366, arXiv:1806.07366.

[DGM98] PAUL DUPUIS, ULF GRENANDER, and MICHAEL I. MILLER. Variational problems on flows of diffeomorphisms for image matching. *Quarterly of Applied Mathematics*, 56(3):587–600, 1998. URL: http://www.jstor.org/stable/43638248.

[HSS08] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, 36(3):1171–1220, 2008.

[JHS+15] Sadeep Jayasumana, Richard Hartley, Mathieu Salzmann, Hongdong Li, and Mehrtash Harandi. Kernel methods on riemannian manifolds with gaussian rbf kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(12):2464–2477, 2015. doi:10.1109/TPAMI.2015.2414422.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL: https://arxiv.org/abs/1412.6980, doi:10.48550/ARXIV.1412.6980.

[Lee06] John M Lee. *Riemannian manifolds: an introduction to curvature*, volume 176. Springer Science & Business Media, 2006.

[PSC16] Nicolas Perrin and Philipp Schlehuber-Caissier. Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems. *Systems Control Letters*, 96:51–59, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0167691116300846, doi:https://doi.org/10.1016/j.sysconle.2016.06.018.

[RLF+20] Muhammad Asif Rana, Anqi Li, Dieter Fox, Byron Boots, Fabio Ramos, and Nathan D. Ratliff. Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems. *CoRR*, abs/2005.13143, 2020. URL: https://arxiv.org/abs/2005.13143, arXiv:2005.13143.

[RR07] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL: https://proceedings.neurips.cc/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf.

[UGTP20] Julen Urain, Michele Ginesi, Davide Tateo, and Jan Peters. Imitationflow: Learning deep stable stochastic dynamic systems by normalizing flows. *CoRR*, abs/2010.13129, 2020. URL: https://arxiv.org/abs/2010.13129, arXiv:2010.13129.

License