Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Alexander Johan Arntzen

Normalizing flows and deep Q-networks for shape analysis

Master's thesis in Applied Physics and Mathematics
Supervisor: Elena Celledoni

July 2022

NTNU
Norwegian University of
Science and Technology

Alexander Johan Arntzen

Normalizing flows and deep Q-networks for shape analysis

Master's thesis in Applied Physics and Mathematics
Supervisor: Elena Celledoni
July 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis considers two different approaches for analyzing curves. In the first approach, we apply deep Q-networks as a method to find an optimal reparametrization between two curves. This reparametrization is necessary when comparing curves in the context of shape analysis. The resulting optimization procedure has a linear time complexity but does not always converge.

Secondly, we use normalizing flows as a way to cluster and interpolate data. We propose a clustering method based on normalizing flows and the expectation-maximization algorithm. Using this clustering method on motion capture data we cluster walking and running motions perfectly into separate clusters. We also test interpolation with normalizing flows on motion capture data. The interpolation method produces new data with a high estimated probability, we observe no significant visual difference from linear interpolation.

Sammendrag

Denne oppgaven tar for seg to forskjellige tilnærminger for å analysere kurver. I den første tilnærmingen anvender vi dype q-nettverk for å finne en optimal reparametrisering mellom to kurver. Denne reparametriseringen er nødvendig for å sammenligne kurver med metoder fra formanalyse. Den påfølgende optimeringsprosedyren har lineær tidskompleksitet, men konvergerer ikke alltid.

I den andre tilnærmingen, bruker vi normaliserende flyt å interpolere og klyngeanalysere data. Vi foreslår en klyngeanalysemetode basert på normaliserende flyt og forventning-maksimering. Ved bruk av denne klyngeanalysemetoden på data fra bevegelsesopptak klarer vi å gruppere gåbevegelser og løpebevegelser i to separate grupper. Vi anvender også normaliserende flyt til å interpolere data fra bevegelsesopptak. Interpolasjonen genererer nye data som har en høy estimert sannsynlighet, men det er ingen stor visuell forskjell fra lineær interpolasjon.

Preface

This master thesis concludes my studies of applied physics and mathematics at the Norwegian University of Science and Technology, specializing in industrial mathematics.

I would like to thank my Supervisor, Prof. Elena Celledoni, for her great guidance through the research and writing process. The discussions and advice have been essential for this work.

Alexander Johan Arntzen
Trondheim
July 1, 2022

Contents

Chapter 1

Introduction

In this text, we will consider two approaches for comparing curves. The first approach uses the framework of shape analysis. One of the first to define shapes as geometric objects in the quotient space with respect to a transformation was Kendall[38]. He defined shapes as geometric objects modulo rotation and dilatation, with the corresponding pre-shape space defined as the original space the objects belonged to. Kendall mainly worked with shapes as finite collections of points, but today shape analysis has also been extended to the study of shapes as continuous objects on infinite-dimensional Riemannian manifolds. For an overview of the use of different Riemannian metrics, we refer to [4, 53, 55]

For our applications, we will only consider the space of curves with some regularity conditions, using the *square root velocity transform* (SRVT). This transform was first introduced by Srivastava et al. [74] and has been applied to many other problems [47, 37, 50, 46, 49]. Importantly the SRVT and the Riemannian structure that follows has been used to cluster and interpolate curves [22]. Additionally [14] where able to cluster walking, running, and jumping motions using an SRVT adapted for lie groups Furthermore, the SRVT framework has also been used to close open curves [13].

To obtain geodesics and distances on the shape space, we first need to find an optimal reparametrization of the curves. Algorithms for finding this reparametrization can be grouped into two categories; gradient-based algorithms [74, 75, 5] and dynamic programming [70, 5]. One drawback of dynamic programming algorithms is that the algorithms need to iterate over all feasible pairs of the considered discrete times. Thus dynamic programming can be costly for the reparametrization of an interval with many discrete times. As an alternative to the dynamic programming approach we, therefore, formulate the reparametrization problem as a deterministic Markov decision process (MDP). In this framework, we can apply reinforcement learning algorithms. These algorithms only seek to provide approximate solutions without searching in the entire space for solutions. In particular, deep Q-networks[56] is an algorithm that has been successfully applied the MDP for playing games with discrete action spaces. Although, there are alternatives for continuous actions spaces [51, 72]. Therefore we test deep Q-networks to find an optimal reparametrization for curves defined by test examples, and curves generated by motion capture data.

A different perspective on the analysis of curves is through normalizing flows. A normalizing flow is an invertible mapping from observable data to a space equipped with the normal distribution. Tabak and Vanden-Eijnden [78] where the first to construct a flow transporting data to the normal distribution in a way that maximizes the likelihood of the data. A later paper [77], then proposes a method of constructing a normalizing flow by the composition of parametric functions. The parameters of these maps are then selected to maximize the likelihood of the observed data.

The two main problems when constructing normalizing flows is that each of the functions composing the flow needs to be both injective, and enable fast computation of the Jacobian determinant. Dinh, Sohl-Dickstein, and Bengio[19] proposed two popular strategies they call coupling layers, and multiscale architected. Using this flow architecture they demonstrate realistic sampling and interpolation of high-dimensional image data. Since then there have been many different successful flow models, and we refer to [61, 42] for an overview. In our experiments, we use invertible residual networks[16, 7] and continuous normalizing flows [51, 29]. These models do not restrict the structure of the Jacobian and thus permit more expressive models.

Normalizing flows have been successfully applied in the context of temporal data. For instance, normalizing flows have been used to efficiently generate audio samples [58], video samples [45] and controllable human motion [32]. Common to all these approaches is that the generation of new data is predicated on a state that was determined when generating old data, or the old data itself.

Our approach to temporal data, like human motion, is different from the approaches described. We disregard the time component of the data and consider the entire motion as one data point. This allows us to interpolate and cluster sequences of data in the same manner as with data without a temporal component. This strategy is motivated by the success of shape analysis in analyzing unparametrized curves. Moreover, when clustering data we will use the expectation maximization algorithm [18] in a way heavily inspired by Agnelli et al.[1].

In Chapter 2 we introduce shape analysis and define the shape space we will be working with. We will closely follow the explanation used by the specialization project [2, Chapter 2] preceding this thesis. Here we also explain briefly how the shape analysis provides a method for cluster and interpolat-

[0]For the audio generation model [58] this is technically not true, but this model was trained to mimic an autoregressive model for which it is true[59].

ing curves, independent of reparametrization. Moreover, we explain how the SRVT imposes a structure pre-shape space, and how that structure results in an optimization problem when analyzing each pair of curves. Then, since we will be working with human motion data in the next chapter we also explain an extension of the SRVT to curves in Lie groups [13].

In chapter 3 we first introduce the Markov decision process and how it relates to reinforcement learning. We then explain some reinforcement learning optimization algorithms. Thereafter, we show that the reparametrization problem is a Markov decision process, and we can solve it using the algorithms previously described. In this section, we also deliver some small alterations to deep Q-networks that can be made four our problem. Finally, we show the results of experiments comparing the performance of deep Q-networks and dynamic programming.

In Chapter 4 introduce the concept of normalizing flows and comment on their expressiveness. Then we define the Kullback-Leibler divergence and show how a normalizing flow can be found by minimizing this divergence. Then we define the two normalizing flow models we will be working with; invertible residual networks and continuous normalizing flows. Additionally, we explain how normalizing flows can be used to cluster and interpolate data. We then show clustering and interpolation on test examples and data motion capture data. Finally, we finish the chapter discussing these results.

In Chapter 5 we give some concluding remark about the methods used in this thesis. We also provide some ideas for further work in the application of reinforcement learning for reparametrization, and for the application of normalizing flows in the context of shape analysis.

Chapter 2

Shape analysis

2.1 Definitions and motivation

To start the exploration of the curves we will be working with we follow the approach by Turaga et al. [82]. First we define the the space of of all immersions from the interval $I = [0, 1]$ to $\mathbb{R}^d$

$$\mathcal{P} := \mathrm{Imm}(I, \mathbb{R}^d) = \{c \in C^\infty(I, \mathbb{R}^d) : c'(t) \neq 0 \ \forall \ t \in I\}.$$

This space $\mathcal{P}$ is generally called the *pre-shape space*. We will restrict ourselves to the study of open curves. However, analysis of closed curves $\mathrm{Imm}(S^1, \mathbb{R}^d)$ and surfaces $\mathrm{Imm}(I \times I, \mathbb{R}^d)$ has also been investigated with the shape analysis framework [4].

We would like to study curves independent on their parametrization. More precisely, we define the space of all reparametrizations φ as the group of orientation preserving diffeomorphisms of I

$$\mathrm{Diff}^+(I) = \{\varphi \in C^\infty(I, I) : \varphi(0) = 0, \varphi(1) = 1, \varphi'(t) > 0 \ \forall \ t \in I\}.$$

The group action

$$\psi : \mathrm{Diff}^+(I) \times \mathcal{P} \to \mathcal{P}, \quad \psi(\phi, c) \mapsto c \circ \phi$$

also defines the *shape space* $\mathcal{S}$ as the quotient space under this group action,

$$\mathcal{S} := \mathcal{P} \ / \ \mathrm{Diff}^+(I).$$

The space $\mathcal{S}$ is not manifold, but an orbifold [4]. The elements of $\mathcal{S}$ will be called shapes *shapes*.

2.2 Distance on shape space

Distance is a very useful structure that we would like to impose on the shape shape. A distance function (or pseudometric[1]) is a natural measure of sim-

[1]A pseudometric $d : X \times X \to \mathbb{R}^+$ satisfies all the axioms of a metric expect $d(x, y) = 0$ does not imply $x = y$.

ilarity with we could use to cluster shapes independent of parametrization. To find a pseudometric on $d_{\mathcal{S}}$ on $\mathcal{S}$ we start with a pseudometric $\mathcal{P}$ that is *reparametrization invariant.* That is, $d_{\mathcal{P}} : \mathcal{P} \times \mathcal{P} \to (0, \infty)$ has the property that for all $c_1, c_2 \in \mathcal{P}$

$$d_{\mathcal{P}}(c_1, c_2) = d_{\mathcal{P}}(c_1 \circ \varphi, c_2 \circ \varphi) \quad \forall \varphi \in \mathrm{Diff}^+.$$

A reparametrization invariant pseudometric on $\mathcal{P}$ is not enough to define a pseudometric on $\mathcal{S}$ directly. Instead, a pseudometric on $\mathcal{S}$ is given by

$$d_{\mathcal{S}}([c_1], [c_2]) := \inf_{\varphi \in \mathrm{Diff}^+(I)^+} d_{\mathcal{P}}(c_1, c_2 \circ \varphi),$$

where $[c_1]$ and $[c_2]$ are the shapes with representatives c_1 and c_2. Therefore we impose a Riemannian metric on $\mathcal{P}$.

We then require that the Riemannian metric is reparametrization invariant. More specifically, a Riemannian metric G such that for all $c \in \mathcal{P}$ and $h, k \in T_c\mathcal{P} = C^\infty(I, \mathbb{R}^d)$

$$G_c(h, k) = G_{c\circ\varphi}(h \circ \varphi, k \circ \varphi) \quad \forall \varphi \in \mathrm{Diff}^+(I)^+.$$

Moreover, a reparametrization invariant Riemannian metric will induce a reparametrization invariant pseudometric on $\mathcal{P}$ given by the length of the shortest path [2]

$$d_{\mathcal{P}}(c_1, c_2) = \inf_{\substack{\gamma \in C^\infty([0,1], \mathcal{P}) \\ \gamma(0) = c_1 \\ \gamma(1) = c_2}} \int_0^1 \sqrt{G_{\gamma(t)}(\gamma'(t), \gamma'(t))}\, dt. \tag{2.1}$$

Therefore, pseudometric on $\mathcal{S}$ can be constructed by (2.2). Furthermore, if this shortest path exist then it also gives us a natural way to interpolate between curves.

An obvious metric on $\mathcal{P}$ is the L^2 metric given by

$$G_c(h, k) = \int_I \langle h, k \rangle |c'(t)|\, dt,$$

where $|c'(t)|$ is the length of the vector $c'(t)$. However, as shown in [53] and [54] the L^2 metric induces a vanishing distance on shape space $\mathcal{S}$. This means that for any $c_1, c_2 \in \mathcal{P}$ we can construct curves $\gamma \in C^\infty([0,1], \mathcal{P})$ starting at c_1 ending at $c_2 \circ \varphi$ of arbitrary short length. Therefore we conclude that the L^2 metric is useless for comparing shapes, and must find another metric.

[2]If it exists

2.3 The square root velocity transform

We now restrict our attention to the space of curves starting at the identity $\mathcal{P}_* := \{c \in \mathcal{P} : c(0) = 0\}$. A metric on $\mathcal{P}_*$ will then by imposed by transforming the curves with a diffeomorphism to another Riemannian manifold, then compute the corresponding pullback metric on $\mathcal{P}$. A popular such transformation, first introduced in [74], is the *square root velocity transform* (SRVT)

$$R : \mathcal{P} \to C^\infty(I, \mathbb{R}^d \setminus \{0\}), \quad c \mapsto \frac{c'}{\sqrt{|c'|}}. \tag{2.2}$$

The SRVT has several properties which we will not prove. Instead, we refer to [11, 6]. Firstly, the SRVT is invariant under translation and therefore not injective on $\mathcal{P}$. It will however, be a diffeomorphism between $\mathcal{P}_*$ and $C^\infty(I, \mathbb{R}^d \setminus \{0\})$[11][Theorem 1]. Between these spaces, the SRVT also has an inverse of the form

$$R^{-1}(q)(t) = \int_0^t q(\tau)|q(\tau)|\, d\tau,$$

where $|q(\tau)|$ is the length of the vector $q(\tau)$. Furthermore, imposing the L^2 inner product on $(C^\infty(I, \mathbb{R}^d), \langle \cdot, \cdot \rangle_{L^2})$ produces an inner product space. Thus, the SRVT imposes a pullback distance metric

$$d_{\mathcal{P}_*}(c_1, c_2) = \|R(c_1) - R(c_2)\|_{L^2},$$

and a pullback Riemannian metric,

$$G_c(h, k) = \langle T_c R(h), T_c R(k) \rangle_{L^2}$$

on $\mathcal{P}_*$. Moreover, as explained in [13], G will be a *first order Sobolev metric*, defined by the arc length integral

$$G_c(h, k) = \int_I \langle D_s h^\perp, D_s^\perp k \rangle + \frac{1}{4} \langle D_s h, v \rangle \langle D_s k, v \rangle \, ds,$$

where $ds = |c'|\, dt$, $v = D_s c = \frac{c'}{|c'|}$ is the curve of unit length tangents of c, and $D_s h^\perp = D_s h - \langle D_s h, v \rangle v$ is the projection of $D_s h$ onto the space of curves pointwise orthogonal to v. Also, as before, $|c'|$ denotes the pointwise length of c'.

The space $C^\infty(I, \mathbb{R}^d \setminus \{0\})$ is a non-convex subset of a inner product space. Moreover, for a convex subset U of inner product space $(V, \langle ., . \rangle)$, the shortest path between two vectors $v_1, v_2 \in U$ is the line

$$\mu(t) = v_1 t + v_2(1 - t).$$

Therefore, when the square root velocity (SRV) form of two curves $R(c_1), R(c_2)$ belong to the same convex subset of $C^\infty(I, \mathbb{R}^d \setminus \{0\})$, then the shortest path between them will be the line

$$\eta(t) = R(c_2)t + R(c_1)(1-t).$$

Therefore, the geodesic from c_1 to c_2 is given by

$$\gamma(t) = R^{-1}[R(c_2)t + R(c_1)(1-t)],$$

and the length of γ will be the equal to $d_{\mathcal{P}_*(c_1,c_2)}$. For curves which have an SRV form not connected by a line in $C^\infty(I, \mathbb{R}^d \setminus \{0\})$ (e. g. $c_1 = -c_2$) the situation is more complicated. Either the geodesic of the metric G will not exist (i.e there is no γ that making the the infimum a max in Equation (2.1)), or the geodesic distance will not correspond to the distance $d_{\mathcal{P}_*}$.

When $d \geq 3$, we can make lines in $C^\infty(I, \mathbb{R}^d)$ smoothly into paths in $C^\infty(I, \mathbb{R}^d \setminus \{0\})$ by arbitrary small perturbations [6][Section 2.1]. Thus, the distance given by $d_{\mathcal{P}_*}$ will be the geodesic distance of G. However, when $d \leq 2$, these smooth perturbations are not always possible, and $d_{\mathcal{P}_*}$ will not always agree with the geodesic distances induced by G. As a solution for $d \leq 2$, Bruveris [11][Section 2.2] proposes geodesic completions of $\mathcal{P}_*$. Also, for a treaty of geodesic completions of Sobolev metrics in general, we refer to [12].

To extend the Riemannian geometry of $\mathcal{P}$ to the shape space we use the fact that the SRVT has the equivariance property [13]

$$R(c \circ \varphi) = \sqrt{\varphi'} R(c) \circ \varphi \quad \forall\ \varphi \in \mathrm{Diff}^+, c \in \mathcal{P}.$$

Therefore, by [6, Theorem 3.1], the metrics G and $d_{\mathcal{P}_*}$ are reparametrization invariant, and a reparametrization invariant pseudometric can be extended to $\mathcal{S}_* := \{[c] \in \mathcal{S} : c(0) = 0\}$ by (2.2). Importantly, finding the distance and geodesic between two shapes $[c_1], [c_2] \in \mathcal{S}_*$ has thusly been reduced to finding a reparametrization $\varphi \in \mathrm{Diff}^+(I)$ that minimizes

$$\left\| R(c_1) - \sqrt{\varphi'} R(c_2) \circ \varphi \right\|_{L^2}. \tag{2.3}$$

Thus, when analyzing curves we need to find an optimal reparametrization between the curves.

An optimal solution to the reparametrization problem above will not necessarily exist in $\mathrm{Diff}^+(I)$. Examples of a pair of curves with an optimal reparametrization outside $\mathrm{Diff}^+(I)$ can be found in [82, p.11] and [89, Section 3.2]. One problem is that an optimal reparametrization might have vanishing derivatives. Thus, the optimal solution is not guaranteed to be a diffeomorphism. Still, there are metrics on $\mathcal{P}$ for which an optimal diffeomorphism is guaranteed, as shown in [4]. Alternatively, Bruveris [11] expands the search to

$$\overline{\Gamma} = \left\{ \gamma : I \to I \;:\; \begin{array}{l} \gamma \text{ absolutely continuous} \\ \gamma(0) = 0, \gamma(1) = 1, \\ \gamma' \geq 0 \text{ almost everywhere} \end{array} \right\}$$

The existence problem is then remedied by showing that for curves $c_1, c_2 \in C^2(I, \mathbb{R}^d)$ there exists optimal reparametrizations $\varphi_1^*, \varphi_2^* \in \overline{\Gamma}$ that minimizes

$$d_{\mathcal{P}_*}(c_1 \circ \varphi_1, c_2 \circ \varphi_2).$$

However, in this thesis, we will only consider the reparametrization problem given by Equation (2.3).

2.4 Shape analysis on Lie groups

In computer graphics, a common model of representing human motion is *skeletal animation*. This model represents one instance of human motion as a rooted tree, such that each edge represents a bone and each node represents the joint between the bones. Moreover, each node is given a local coordinate system related to its parent node via rotation and translation. Thus, the global position of any node can be computed iteratively by composing the transformations connecting the node to the root. Furthermore, rotation and translation can be represented using the matrix group $SE(3)$. Therefore each instance of human motion will in skeletal animation be given as an element in the *joint space*

$$\mathcal{J} = SE(3)^d,$$

where d is the number of bones in the model.

Furthermore, one second of skeletal animation will be given as a time curve $c \in \mathcal{P}$, where

$$\mathcal{P} := \operatorname{Imm}(I, G) = \{c \in C^\infty(I, G) : c'(t) \neq 0 \ \forall \ t \in I\}.$$

This will be the pre-shape space used when using shape analysis on Lie groups. Additionally, all human bones have fixed lengths. Therefore transformations between all bones can be represented as rotation matrices in $SO(3)$. Thus, the joint space of a rigid human skeleton will be the Lie group $SO(3)^d$.

To analyze curves in this Lie group we use use an extension to the SRVT introduced by Celledoni, Eslitzbichler, and Schmeding [13]. This SRVT is defined by

$$\mathcal{R} : \mathcal{P} \to C^\infty(I, \mathfrak{g} \setminus \{0\}), \quad \mathcal{R} := \frac{\delta^r(c)}{\sqrt{\|\delta^r(c)\|}},$$

where $\mathfrak{g}$ is the Lie algebra of G and δ^r is the *right logarithmic derivative* as defined in [43][p. 404]. For our purposes, we note by [57][p.72] that for subgroups of $GL(n)$ the right logarithmic derivative is given by

$$\delta^r(c)(t) = c'(t)c(t)^{-1},$$

where $c'(t)$ is the matrix given by the time derivative of c at t, and $c(t)^{-1}$ is the matrix inverse of $c(t)$. Futhermore, $\mathcal{R}$ is translation invariant [13][Lemam 3.6], meaning that for the right translation given by $R_{g_1}(g_2) = g_1 \cdot g_2$, we have

$$\mathcal{R}(c) = \mathcal{R}(R_g \circ c) \quad \forall\ g \in G, c \in \mathcal{P}.$$

Thus, we define as before the space of curves staring at the identity[3] $\mathcal{P}_* = \{c \in \mathcal{P} : c(0) = e\}$.

Celledoni, Eslitzbichler, and Schmeding also proved that $\mathcal{R}$ has many of the same properties as the SRVT defined in [74]. Firstly, by [13][Theorem 3.16], for dim $\mathfrak{g} > 2$, the function

$$d_{\mathcal{P}_*}(c_1, c_2) = \|R(c_1) - R(c_2)\|_{L^2},$$

will be a metric on $\mathcal{P}_*$ and corresponds to the geodesic distance induced by the elastic metric given in [13][Theorem 3.11]. Moreover, since $\mathcal{R}$ is reparametrization equivariant, by [13][Lemma 3.6], then $d_{\mathcal{P}_*}$ defines a well defined distance function on $\mathcal{S}\ /\ \mathrm{Diff}^+(I)$ by (2.2).

In a practical setting the continuous curve c may only be known at discrete times $(t_i)_{i=0}^n$. Then a continuos curve $\bar{c}$ can be constructed by interpolating along geodesics in $SO(3)$ between each $c(t_i)$ and $c(t_{i+1})$, $0 \leq i \leq n-1$ [71]. Then, by [13][p. 23], the SRV form of $\bar{c}$ will be the piecewise constant curve

$$\bar{q}(t) = \mathbb{1}_{[t_i,t_{i+1})} \frac{\eta_i}{\sqrt{\|\eta_i\|}}, \tag{2.4}$$

where

$$\eta_i = \frac{\log\left(c_{i+1}c_i^T\right)}{t_{i+1} - t_i},$$

and log is the Lie group logarithm.

[3]For Lie groups $GL(N)$ the identity e is the matrix identity I_n. However for $\mathbb{R}^n$ as a Lie group the identity is 0.

Chapter 3

Reparametrization with deep Q-networks

When analyzing unparametrized curves in the shape analysis framework, one has to solve an optimization problem for each unordered pair of considered curves. Specifically, for a pair of curves $c_1, c_2 \in \mathrm{Imm}(I, \mathbb{R}^d)$ we need to find a diffeomorphism $\varphi \in \mathrm{Diff}^+(I)$ that minimizes Equation (2.3). Since this problem must be solved for each pair of curves, the optimization procedure must be efficient. Finding an optimal reparametrization has previously been solved using a dynamic programming algorithm, but this algorithm comes with a high computation cost. This chapter shows that the reparametrization problem is a deterministic Markov decision process (MDP). Additionally, we explore different reinforcement learning algorithms that find an optimal policy for MDP.

3.1 Markov decision processes

Before describing dynamic programming as a method to find an optimal reparametrization, we first introduce the formalism of a Markov decision process (MDP). A dynamic programming method can be applied directly to problems without the MDP framework. However, reinforcement learning algorithms like deep Q-learning are created to solve MDPs.

There are many versions of MDPs [65, Chapter 1], but we will use a deterministic version from [23, Chapter 3]. Essential to all MDPs is the existence of states, actions, and rewards. An MDP models a system where an actor moves between states while taking actions that determine the reward received for each movement. The MDP framework can be used to model stochastic movements between states. However, we restrict ourselves to deterministic MDPs. Since we are interested in minimizing Equation (2.3) rather than maximizing rewards, we will modify the definition of MDP to fit the minimization goal.

Definition 3.1.1 (Deterministic Markov decision process)**.** *A deterministic Markov decision process is tuple $(\mathcal{S}, \mathcal{A}, T, C)$. Where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function mapping previous state and actions to new states, and $C : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$ is the cost received when for each state-action pair.*

Moreover, many optimization algorithms require that the state space S and action space $\mathcal{A}$ are finite sets. However, some algorithms work on continuous action and state spaces [88, Chapter 7][51, 72].

Remark. *When working with a Markov process [79] instead of a deterministic process, we need to make two modifications of the deterministic MDP. Firstly, The transition function T is now a function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$, such that $T(s, a, .)$ is the probability distribution of the next state given previous state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$. Secondly, the cost function $C : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$ is now a function of both the previous and successive state.*

When working with Markov decision processes it is useful to define policies and value functions. A *policy* is a mapping

$$\pi : \mathcal{S} \to \mathcal{A},$$

that determines the action taken for each state $s \in \mathcal{S}$. The associated *value function* for π is defined by the recurrence relation

$$V^\pi : \mathcal{S} \to \mathbb{R}^+ \quad V^\pi(s) = \gamma V^\pi(s') + C(s, a), \tag{3.1}$$

where $s' = T(s, a)$ is the next step when taking action $a = \pi(s)$ given by policy π, and $\gamma \in [0, 1]$ is a constant that discounts future value. If no discounting is done $\gamma = 1$. Then for V^π in Equation (3.1) to be well defined we also need to define the value of $V^\pi(s_{\text{end}})$ for one or more end states s_{end}. A Markov decision process with end states is called a *finite horizon* process. Furthermore, if future value is not discounted, then the transformation function T must not allow revisiting the same state; which in our case will be true. Finally, we define the *optimal value function*

$$V^* : \mathcal{S} \to \mathbb{R}^+ \quad V^*(s) = \min_{\pi \in \Pi} V^\pi(s),$$

where Π is the space of all policies $\Pi = \{\pi : \mathcal{S} \to \mathcal{A}\}$.

An important property of the optimal value function V^* is that it is the only function that satisfies the Bellman equation

$$V(s) = \min_{a \in \mathcal{A}} [\gamma V(T(s, a)) + R(s, a)], \tag{3.2}$$

for all $s \in \mathcal{S}$. Thus if V^* is known, the value of an *optimal policy*, a policy π^* such that $V^{\pi^*} = V^*$, can be computed by

$$\pi^*(s) = \operatorname*{argmin}_{a \in \mathcal{A}} \left[\gamma V^*(T(s, a)) + C(s, a)\right]. \tag{3.3}$$

For a value function V this policy is called the greedy policy. The greedy policy can also be generalized to include multiple steps. As defined by Efroni et al. [21] a h-greedy policy with respect to a value function V is a policy π_h so that for a state s_0

$$\pi_h(s_0) = \operatorname*{argmin}_{a_0 \in \mathcal{A}} \min_{a_1,\dots,a_{h-1} \in \mathcal{A}} \left[\sum_{t=0}^{h-1} C(s_t, a_t) + \gamma^h V(s_h) \right] \quad s_{i+1} = T(s_i, a_i). \tag{3.4}$$

Another central value function in the MDP model is the optimal action-value function Q. For deterministic MDPs the optimal action-value function Q^* can be defined as the function that satisfies the recurrence relation

$$Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+ \quad Q^*(s, a) = \min_{a' \in \mathcal{A}} \gamma Q^*(T(s, a), a') + C(s, a), \tag{3.5}$$

for all non final states $s \in S$ and actions $a \in \mathcal{A}$. For final states s_{end} we define $Q^*(s_{\text{end}}, a) = V^*(s_{\text{end}})$. In other words, $Q^*(s, a)$ is the final cost the agent achieves if at state s the agent takes action a, then choosing the optimal action at all later states. Therefore, if Q^* satisfies (3.5) then an optimal value function will be given by

$$V^*(s) = \min_{a \in \mathcal{A}} Q^*(s, a).$$

The Q-function also defines a greedy strategy similarly to Equation (3.3). Substituting

$$Q^*(s, a) = C(s, a) + \gamma V^*(T(s, a)), \tag{3.6}$$

into Equation (3.3) yields the greedy policy π with respect ot an action-value function Q

$$\pi_0(s) = \operatorname*{argmin}_{a \in \mathcal{A}} Q(s, a). \tag{3.7}$$

Similarly, the h-greedy policy for an action-value function Q and state s_0 is

$$\begin{gathered} \pi_h(s_0) = \operatorname*{argmin}_{a_0 \in \mathcal{A}} \min_{a_1,\dots,a_{h-2} \in \mathcal{A}} \left[\sum_{t=0}^{h-1} C(s_t, a_t) + \gamma^{h-1} Q(s_{h-1}, a_{h-1}) \right] \\ s_{i+1} = T(s_i, a_i). \end{gathered} \tag{3.8}$$

3.2 Value iteration

Value iteration is a classical way of computing V^* for discrete action and state spaces and was developed by Bellman [8]. Together with policy iteration [34], it is one of the two central dynamic programming algorithms for solving discrete MDPs. They are both guaranteed to converge to an optimal policy for finite horizon problems.

Finite horizon value iteration assumes two reasonable properties about the MDP it is applied to. Firstly that each state $s_t \in \mathcal{S}$ is indexed with a known time $t \in \mathcal{I}$. For each time t we will denote the space of states index by time t as S_t. Secondly, for temporal indexing to have any meaning we require that if $t' \geq t$ then for $s_t \in S_t$ and $s_{t'} \in S_{t'}$ there is no action $a \in \mathcal{A}$ such that $s_t = T(s_{t'}, a)$. Using this temporal indexing, value iteration computes the value of the optimal value function V^* backwards in time as described in Algorithm 1. Since value iteration computes the values of V^* exactly for each state it iterates over, the time complexity of the algorithm will be $\Theta(|\mathcal{S}||\mathcal{A}|)$.

Algorithm 1 Finite horizon value iteration

```
initialize $V(s_{t_N}) = 0 \ \forall s_{t_N} \in S_{t_N}$
for $i \leftarrow N - 1$ to 1 do
    for all $s_t \in S_{t_i}$ do
        $V(s_t) \leftarrow \min_{a \in \mathcal{A}} [V(T(s_t, a)) + C(s_t, a)]$
    end for
end for
```

3.3 Q-learning

Q-learning is an algorithm in a family of algorithms called *reinforcement learning*. Algorithms in the reinforcement learning framework are made for the general problem where an agent learns how to act in an environment by receiving rewards [88, Chapter 1]. However, specific reinforcement learning algorithms require more concrete assumptions about the agent and the environment. In particular, a MDP is the most common assumption, but other models are also possible [73]. In contrast to dynamic programming algorithms, reinforcement learning algorithms generally need not assume any knowledge about the MDP itself; in other words, they are *model-free*. In addition, reinforcement learning algorithms do not guarantee convergence to an optimal solution to their problem. Therefore, they do not need to visit all states to arrive at a solution like dynamic programming. Instead, these algorithms provide an approximate solution to the problem.

The Q-learning algorithm [86] is a *temporal-difference (TD)* method for estimating the optimal action-value function Q^*. Common to all TD algorithms is that they sequentially update estimates based on other estimates and observing how the environment reacts to actions [76, Chapter 6.]. In particular, Q-learning updates it estimates of Q^* by moving between states and selecting actions based on the current estimate Q. One popular policy for selecting actions is an ϵ-*greedy* policy. An ϵ-greedy is a policy

$$\pi_\epsilon(s) = \begin{cases} a \sim \mathcal{U}(A), & \text{with probability } \epsilon \\ \pi^*(s), & \text{else,} \end{cases}$$

where π^* is a greedy policy. Based on the previous state-action pair (s, a) the Q-learning algorithm then updates it current estimate of the table Q by

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left[C(s,a) + \gamma \min_{a' \in \mathcal{A}} Q(T(s,a), a') \right],$$

where α is a parameter $0 < \alpha < 1$. Finally, we note that Q-learning does converge to a solution with an optimal greedy policy for discrete MDP as long as all states-action pares are repeatedly sampled [85].

3.4 Deep Q-learning

After Watkins introduced Q-learning, there have been multiple adaptions of the algorithms using function approximations Q_θ as an estimate for Q^* [23, Chapter 4.]. Central to the algorithms we will explore is the use of *replay memory* [52]. Instead of updating Q_θ based on the last experiences $(s, a, C(s,a), T(s,a))$, the last N_{replay} are stored in database called replay memory. Then Q_θ is updated based on a mini-batch sample from the replay memory. Mini-batches reduce each update's variance of Q while covering a larger portion of the state and action space.

Riedmiller [66] parametrized the Q_θ-function as neural networks. However, instead of taking state-action pairs as inputs, the network inputs the state then computes a separate output for each action in the action space. This approach enables efficient computation of the maximum of all actions and continuous state spaces. Training the network then amounts to sampling mini-batches $(s_k, a_k, c_k, s')_{k=1}^{N_{\text{batch}}}$ from replay memory and minimizing the cost function

$$\mathcal{L}(\theta) = \sum_{k=0}^{N_{\text{batch}}} (Q_\theta(s_k, a_k) - Y_k)^2, \tag{3.9}$$

where Y_k is the target

$$Y_k = c(s_k, a_k) + \gamma \min_{a' \in \mathcal{A}} Q_{\theta_i}(s_k', a') \tag{3.10}$$

One problem with using nonlinear function approximators in Q-learning is that we are not guaranteed to converge to a final solution [81, 66, 9]. To combat this possible divergence Mnih et al. [56] separated generation of targets and optimization into two different neural networks Q_θ and $\hat{Q}_{\theta^-}$, which they call a deep Q-network (DQN). More specifically, the target function $\hat{Q}_{\theta^-}$ is used to compute the targets Y_k by Equation (3.10) and Q_θ is optimized with respect to the cost function $\mathcal{L}$ (3.9). Then the target $\hat{Q}$ is updated by $\theta^- \leftarrow \theta$ every C iterations. Using a deep Q-network, Mnih et al. then demonstrated experimentally increased stability on Atari games. A version their algorithm for deterministic MDPs is described in Algorithm 2.

Algorithm 2 Deterministic deep Q learning

```
Require: Function $Q_\theta$ with parameters $\theta$
Require: Function $\hat{Q}_{\theta-1}$ with parameters $\theta^- = \theta$
Require: Initialized replay memory $\mathcal{D}$
  for episode ← 1 to $N$ do
      $s \leftarrow s_{\text{start}}$
      while $s \neq s_{\text{end}}$ do
          Select action $a$ according to some strategy (e.g. $\epsilon$-greedy)
          $s', c = T(s,a), C(s,a)$
          Store experience $(s,a,c,s')$ in replay memory $\mathcal{D}$
          Sample $N_{\text{batch}}$ samples $(s_k, a_k, c_k, s'_k)$ from $\mathcal{D}$
          for $k \leftarrow 1$ to $N_{\text{batch}}$ do
              if $s'_k = s_{\text{end}}$ then
                  $Y_k \leftarrow c_k$
              else
                  $Y_k \leftarrow c_k + \gamma \max_{a\in\mathcal{A}} \hat{Q}_{\theta^-}(s'_k, a_k)$
              end if
          end for
          Do one optimization step of $\mathcal{L}(\theta)$ given by (3.9) w.r.t $\theta$
          Every $C$ steps $\theta^- \leftarrow \theta$
          $s \leftarrow s'$
      end while
  end for
```

3.5 Dynamic programming principle for reparametrization

Before formulating reparametrization as an MDP we show that finding the optimal reparametrization fulfills a dynamic programming principle (DPP) [8, Chapter 3]. We begin by defining the optimal value function

$$V^*(t_0, x_0) := \inf_{\substack{\varphi\in\text{Diff}^+(I)\\ \varphi(t_0)=x_0}} \int_{t_0}^{1} |R(c_1) - R(c_2\circ\varphi)|^2\, dt,$$

where R is the SRV transform, $t_0, x_0 \in [0,1]$ and $I = [0,1]$. Moreover, V^* is a value function for optimal reparametrization since by definition we have

$$V^*(0,0) = d_{\mathcal{P}_*}(c_1, c_2).$$

Now we expanding V^* for a intermediate time $t' \in [t_0, 1]$ we get the following dynamic programming principle

$$V^*(t_0, x_0) = \inf_{\substack{\varphi \in \mathrm{Diff}^+(I) \\ \varphi(t_0)=x_0}} \left[V^*(t'\varphi(t')) + \int_{t_0}^{t'} |R(c_1) - R(c_2 \circ \varphi)|^2 \, dt \right]. \quad (3.11)$$

One possible strategy would be to apply a continuous time dynamic programming algorithm directly on Equation (3.11), however this approach will not be used in this text.

3.6 Reparametrization as a MDP

To formulate reparametrization as a discrete MDP we follow a discretization method inspired by Sebastian, Klein, and Kimia [70]. They do not use the MDP formalism but use dynamic programming to solve a discretized version of Equation (3.11). The method first discretizes the interval I into $\mathcal{I} = \{\tau_0, \dots, \tau_N\}$. Then we define our state space as $\mathcal{S} = \mathcal{I} \times \mathcal{I}$, and our action space as $\mathcal{A} = \mathbb{Z}_d \times \mathbb{Z}_d$, where $d \leq N$ is a natural number called *depth* of the action space. Then the transition function T has the natural form

$$T((\tau_i, \tau_j), (\Delta i, \Delta j)) := (\tau_{\min(N, i+\Delta i)}, \tau_{\min(N, j+\Delta j)}).$$

Now, each path trough the grid $\mathcal{S}$ starting at $s_1 = (0, 0)$ and ending at $s_{\text{end}} = (1, 1)$ will represent a reparametrization that linearly interpolates the two points representing successive states. More precisely, for a sequence of states $S = (t_1, x_1), (t_2, x_2), ..., (t_M, t_M)$ the reparametrization φ_S associated with the sequence S is defined as

$$\varphi_S(t) = \sum_{i=1}^{M-1} \mathbb{1}_{[t_i, t_{i+1}]} l_{s_i, s_{i+1}}(t),$$

where $l_{s_i, s_{i+1}}$ is the affine function interpolating points s_i, s_{i+1} given by

$$l_{s_i, s_{i+1}}(t) = \frac{x_{i+1} - x_i}{t_{i+1} - t_i}(t - t_i) + x_i.$$

Similarly, a policy π will determine a unique path $S^\pi = s_{\text{start}}, s_2^\pi, s_3^\pi, ..., s_{\text{end}}$ iteratively by

$$s_{i+1}^\pi = T(s_i^\pi, \pi(s_i^\pi)) \quad \text{if} \quad s_i \neq s_{\text{end}}.$$

Thus each policy also defines a piecewise linear reparametrization $\varphi_\pi = \varphi_{S^\pi}$.

Furthermore, if we define our cost function C as

$$C((t, x), a) = \int_t^{t'} \left| R(c_1) - R(c_2 \circ \varphi_{(t,x),(t',x')}) \right|^2 d\tau, \quad T((x, t), a) = (t', x'),$$

and $V(s_{end}) = 0$, then

$$V^\pi(s_{\text{start}}) = \int_0^1 |R(c_1) - R(c_2 \circ \varphi_\pi)|^2 \, d\tau.$$

Thus minimizing $V^\pi(s_{\text{start}})$ is equivalent to minimizing (2.3) in restricted search space $\{\varphi_\pi | \pi \in \Pi\}$. Moreover if we know the optimal value function V^*, then we can find an optimal reparametrization via Equation (3.3).

3.7 Reparametrization with deep Q-networks

Formulating the reparametrization problem as an MDP we enable the use of various reinforcement learning algorithms. The algorithm proposed by Sebastian, Klein, and Kimia [70] can be seen as an adaption of value iteration on reparametrization as a MDP. The required time indexing is fulfilled by using the original time of the curve. Although, value iteration is guaranteed to arrive at an optimal solution it has time complexity $\Theta(|\mathcal{S}||\mathcal{A}|)$. As previously explained, another algorithm is deep Q-learning. Using DQN we can find an approximately optimal reparametrization with time complexity $\Theta(|\mathcal{I}|EN_{\text{batch}})$, where E is the predetermined number of episodes used and N_{batch} is the mini-batch size. However, in opposition to value iteration, there are no guarantees for convergence to an optimal policy π^*.

In the reparametrization problem, we can compute C and T. Therefore, we modify the deep Q learning algorithm to use this knowledge about the MDP. Instead of choosing actions with an ϵ-greedy policy, we choose actions with an ϵ-2-greedy policy. More specifically, for a state s we choose the an action according to

$$\pi_\epsilon(s) = \begin{cases} a \sim \mathcal{U}(A), & \text{with probability } \epsilon \\ \pi_2(s), & \text{else,} \end{cases}$$

where π_2 is the h-greedy policy (3.8) with respect $\hat{Q}_\theta^-$ and $h = 2$. This adaption does however, come with an increased computation cost and a time complexity $\Theta(|\mathcal{I}|E(N_{\text{batch}} + |\mathcal{A}|))$.

3.8 Experiments

In this section we compare deep Q-networks (DQN) with the value iteration algorithm [70] to find an optimal reparametrization for three different pair of curves c_1, c_2 and their SRV form q and r. We also compare our results with a greedy strategy; an algorithm that solves a problem using the locally optimal strategy. For our deterministic MDP, a greedy strategy is the strategy that for each state $s \in \mathcal{S}$ selects the action

$$a = \underset{a \in \mathcal{A}}{\operatorname{argmin}}\, C(s, a).$$

For all three curve pairs, we discretize the problem to a discretize MDP as outlined in Section 3.6. Then optimization with (DQN) is performed using the SRV transformed curves q, r with different parameters. The parameters we varied were: the use of the 2-greedy policy, the values of $\epsilon, C, N_{\text{batch}}, N_{replay}$ and the parameters used for optimization of $\mathcal{J}(\theta)$. We also test different discrete times N, but we will use the same action space with a depth of $d = 4$. Finally, for each algorithm's computed policy, we compare the total received cost $V^{\pi}(0,0)$. This value is the approximated distance given by Equation (2.3). For the (DQN) iterations, we will also compare the value of $\mathcal{L}(\theta)$, which we will denote as Q loss.

3.8.1 Experiment 1

In this experiment we compare two cures that are reparametrizations of each other. The SRV form of the two curves r, q are shown in Figure 3.1 and defined by

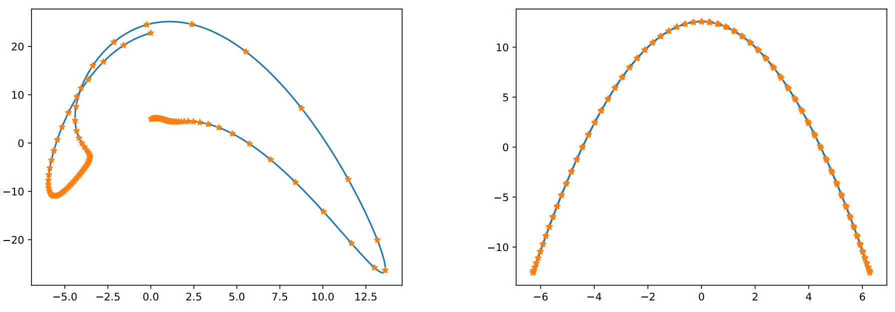
$$r(t) = \pi[-2\sin(2\pi t), 4\cos(4\pi t)]^T,$$

and its reparametrization $q = \sqrt{\psi'}q \circ \psi$, where

$$\psi(t) = \frac{\log(20t+1)}{2\log(21)} + \frac{\tanh(20(t-0.5))}{4\tanh(10)}.$$

These curves were also considered in [67, Section 4.2.6].

The curves where then linearly interlated in $N = 32$ equidistant times.



(a) q (b) r

Figure 3.1: The trajectories of curves q and r defined in Experiment 1

The results of comparing algorithms on the curves q, r with different parameters for $N = 32$ discrete times are shown in Figure 3.2. For a comparison of the best solution using DQN compared to value iteration, see Figure 3.3. Additionally, a comparison of the convergence history for the best and worst runs using DQN is shown in Figure 3.4. We also performed experiments with different N. The resulting cost for each run is shown in Figure 3.5.

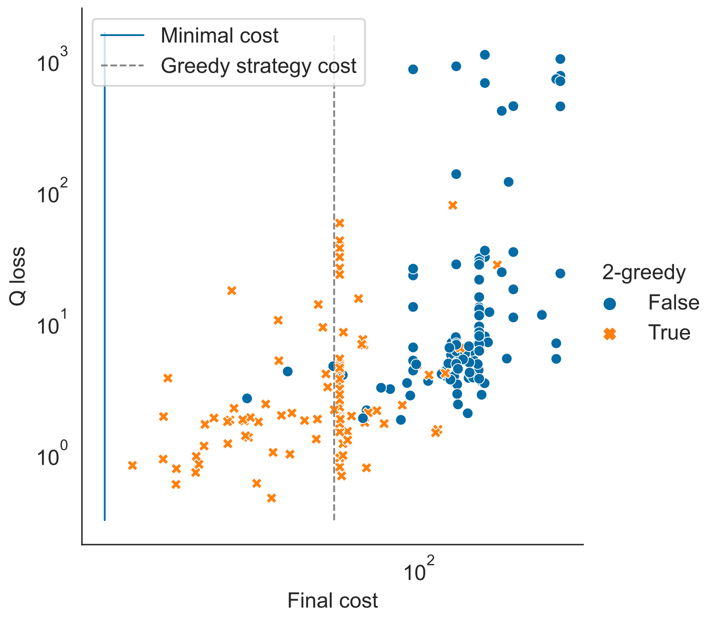
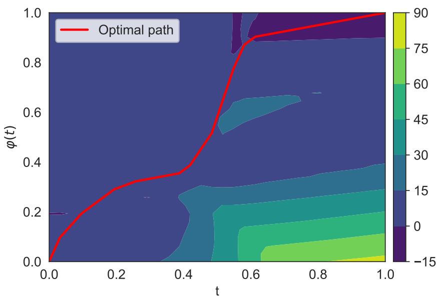
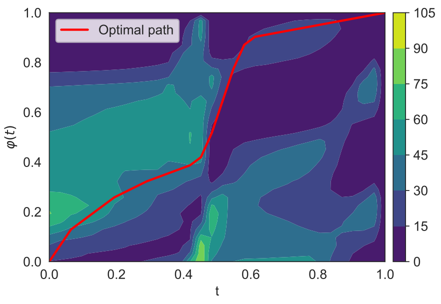


Figure 3.2: The result using DQN in Experiment 1 with $N = 32$ discrete times for 144 different runs. For each run we compare the final cost and the final Q-loss. 42 of out the 216 models performed better than the greedy strategy.

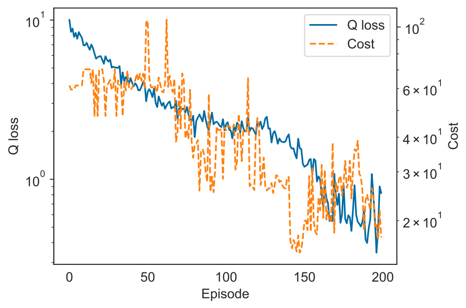


(a) DQN

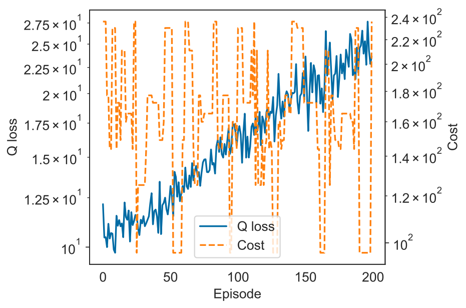


(b) Value iteration

Figure 3.3: Optimal path and value function for the best deep Q learning result, and the result of value iteration. This figure shows the result of the iteration with the lowest final cost for Experiment 1 with $N = 32$ discrete times. The color corresponds to he value function V at each point in figure.



(a) Best training history



(b) Worst training history

Figure 3.4: Figure shows how the Q loss and associated total cost evolves with each episode of the DQN algorithm. The left(right) figure shows the history of the iteration concluding in the best (worst) final cost for Experiment 1 with $N = 32$ discrete times.

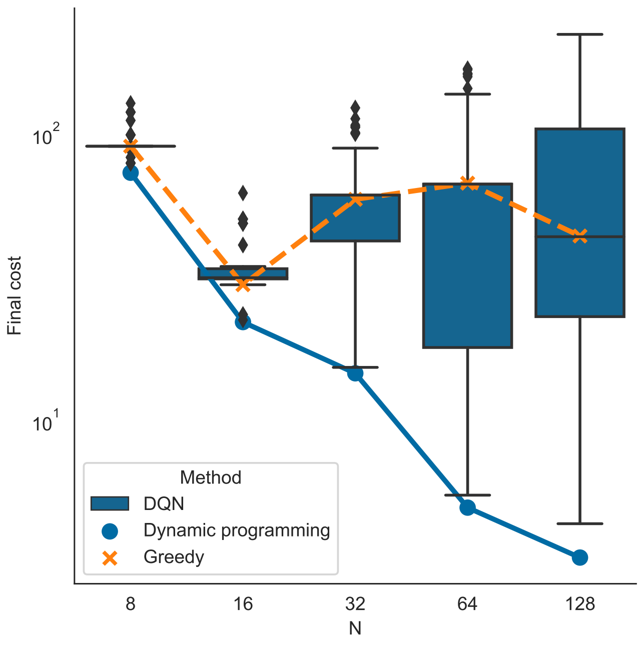


Figure 3.5: The final cost (computed distance) with DQN compared to dynamic programming and the greedy strategy for different number N of discrete times. Deep Q-learning was performed 108 times for each N with 2-greedy strategy, but other parameters where varied.

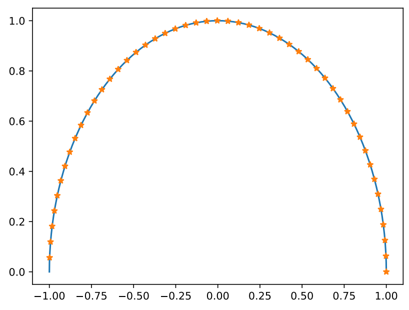
3.8.2 Experiment 2

In this experiment we considered two curves that are not reparametrizations of each other. Their SRV forms are shown in Figure 3.6 and given by

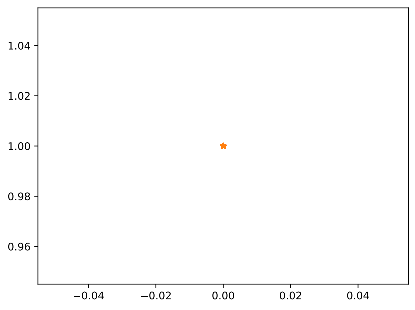
$$q(t) = [\cos(t), \sin(t)], \quad r(t) = [0, 1].$$

The corresponding optimal reparametrization problem of the two curves has an analytical solution computed in [89, A.1] and given by

$$\varphi(t) = t - \frac{\sin(2\pi t)}{2\pi}.$$



(a) q



(b) r

Figure 3.6: The trajectories of curves q and r defined in Experiment 2

The results of comparing algorithms on the curves q, r with different parameters for $N = 32$ discrete times are shown in Figure 3.2. The parameters we varied were: the use of the 2-greedy policy, the values of $\epsilon, N_{\text{batch}}$, and the optimization parameters. The results of these experiments using $N = 32$ discrete times are shown in Figure 3.7. For a comparison of the best solution using deep Q-learning compared to value iteration, see Figure 3.8. Additionally, a comparison of the convergence history for the best and worst runs using deep Q-learning is shown in Figure 3.9.

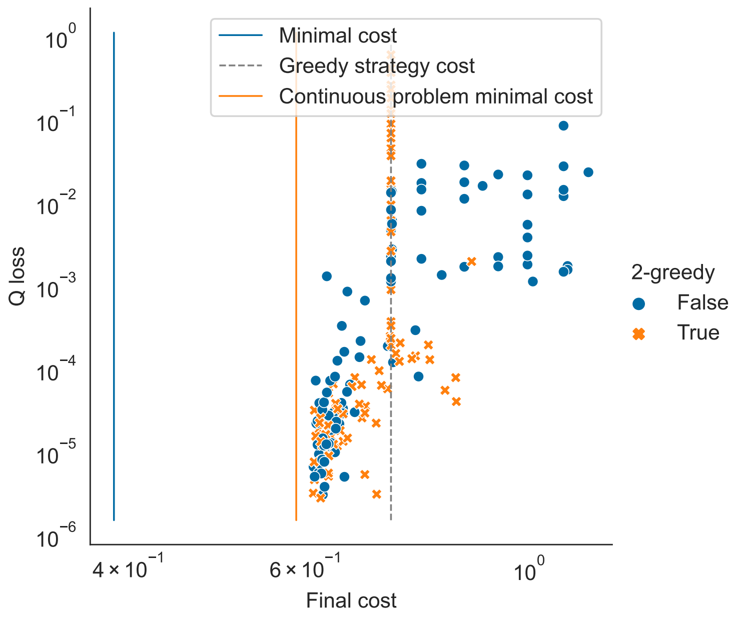
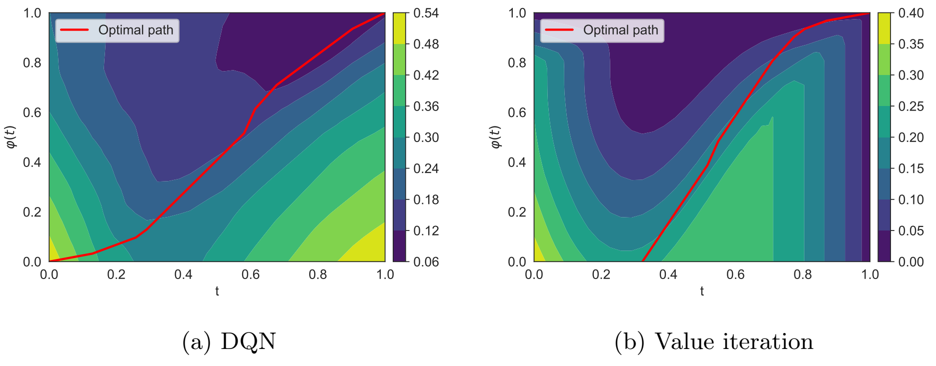


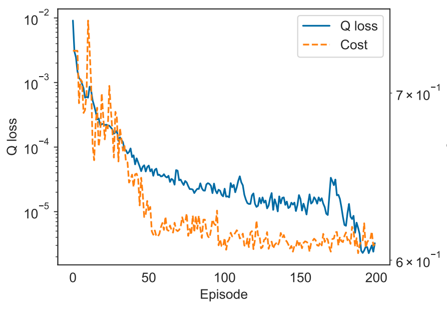
Figure 3.7: The result using DQN in Experiment 2 with $N = 32$ discrete times for 216 different runs. For each run we compare the final cost and the final Q-loss. 123 out of the 216 runs performed better than the greedy strategy.



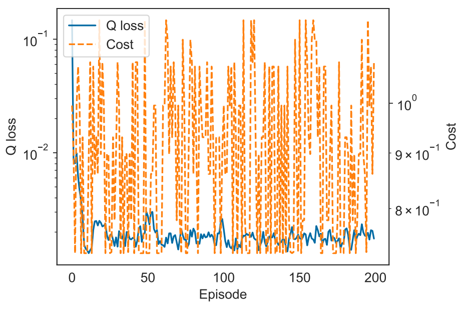
(a) DQN

(b) Value iteration

Figure 3.8: Optimal path and value function for the best DQN result, and the result of value iteration. This figure shows the result of the iteration with the lowest final cost for Experiment 2 with $N = 32$ discrete times. The color corresponds to he value function V at each point in figure.



(a) Best training history



(b) Worst training history

Figure 3.9: Figure shows how the Q loss and associated total cost evolves with each episode of the DQN algorithm. The left(right) figure shows the history of the iteration concluding in the best (worst) final cost for Experiment 2 with $N = 32$ discrete times.

3.8.3 Experiment 3

In this experiment, we consider two discrete curves generated by motion capture data from the *CMU Graphics Lab Motion Capture Database* [48]. The first curve is a brisk walk generated by Subject 7, and the second is a walk with a turn generated by Subject 38. The curves are then transformed using (2.4) into their a SRV forms q, r with $N = 78$ discrete times.

The results of comparing algorithms on the curves q, r with different parameters are shown in Figure 3.10. For a comparison of the best solution using deep Q-learning compared to value iteration, see Figure 3.11. Additionally, a comparison of the convergence history for the best and worst runs using deep Q-learning is shown in Figure 3.12.

3.9 Discussion

As seen in figures 3.2, 3.7 and 3.10 a decrease in in the final Q loss $\mathcal{L}(\theta)$ does in general lead to a lover total cost. Similarly, as seen in figures 3.4, 3.9 and 3.12 the most successful iterations of DQN lower the Q-loss and the total cost at at the same time, while unsuccessful iterations fail to converge to any value for Q. Moreover, as seen in Figure 3.5 DQN is able to make successively better reparametrizations as N increases. This indicates that using function approximators like DQN to find a good reparametrization is possible.

On the other hand, the use of DQN for the reparametrization problem does include a lot of instability as a solution strategy. In all our experiments, only around half of the runs using DQN resulted in a reparametrization better than the greedy policy. Including the ϵ-2-greedy policy improves stability, but does not guarantee a better solution than the greedy strategy. It is possible that picking better parameters would lead to more stable results, but we have not been successful in finding a set of parameters that consistently outperforms the greedy strategy for different curve pairs and problem sizes N. In addition, even through the time complexity fot the DQN algorithm is linear in the size of state space, the runtime of the algorithm is noticeably longer than dynamic programming even for problems as large as $N = 120$. Therefore, a good approximation using dynamic programming can more reliably be achieved by restricting the problem size.

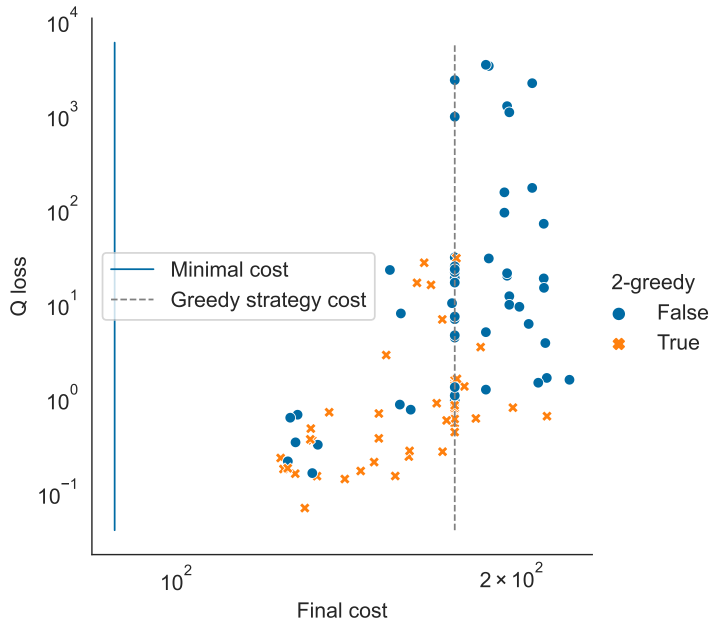
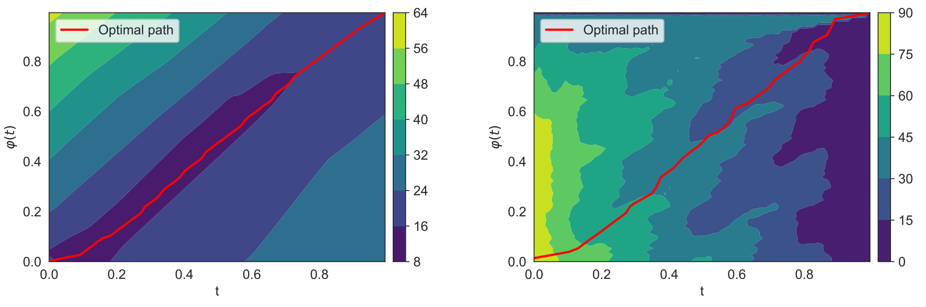


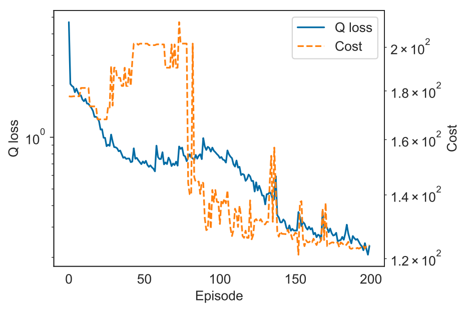
Figure 3.10: The result using (DQN) in Experiment 3 for 108 different runs. For each run we compare the final cost and the final Q-loss. Out of the 108 runs 59 performed better than the greedy strategy.



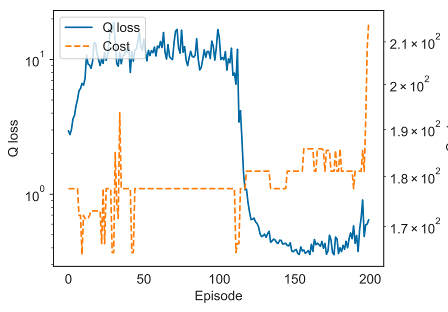
(a) DQN

(b) Value iteration

Figure 3.11: Optimal path and value function for the best deep Q learning result, and the result of value iteration. This figure shows the result of the iteration with the lowest final cost for Experiment 3. The color corresponds to he value function V at each point in figure.



(a) Best training history



(b) Worst training history

Figure 3.12: Figure shows how the Q loss and associated total cost evolves with each episode of the DQN algorithm. The left(right) figure shows the history of the iteration concluding in the best (worst) final cost for Experiment 3.

Chapter 4

Normalizing flows

Normalizing flow are generative models for estimating the unknown probability distribution of a random variable X using a known random variable Z. More formally, we have a probability space $(\mathcal{X}, \mathcal{A}_{\mathcal{X}}, P^X)$ where $P^X = \text{law}(X)$ is the unknown probability distribution of X. The goal is to estimate P^X by introducing a known probability space $(\mathcal{Z}, \mathcal{A}_{\mathcal{Z}}, P^Z)$ and a *latent* random variable Z such that $\text{law}(Z) = P^Z$. Then, the distribution P^Z is transformed into P^X by a measurable function $T : \mathcal{Z}, \to \mathcal{X}$ such that $T_* P^Z = P^X$. Here $T_* P^Z$ denotes the pushforward probability measure defined by

$$T_* P^Z(A) = P^Z(T^{-1}(A)) \quad \forall \quad A \in \mathcal{A}_{\mathcal{X}}.$$

Normalizing flows usually rely on a more specific set of assumptions than mappings between probability spaces in general. Firstly, it is assumed that T is invertible. Moreover, we will only work with $\mathcal{Z} = \mathcal{X} = \mathbb{R}^d$ and $\mathcal{A}_{\mathcal{Z}} = \mathcal{A}_{\mathcal{X}} = \mathcal{B}(\mathbb{R}^d)$, the Borel σ-algebra on $\mathbb{R}^d$. However, normalizing flows have been applied to discrete spaces [33] [80], and Riemannian manifolds [24][84]. We will also need to assume that P^Z is absolutely continuous with respect to the Lebesgue measure so that the probability density p_X of P^X exists. Finally, we let the latent variable Z be the standard gaussian variable $\mathcal{N}(0, I_n)$ and structure T as a composition of multiple transformations $T_N \circ ... \circ T_2 \circ T_1$. Thus, we arrive at the reason for the name *normalizing flow*; the function T^{-1} defines a flow that transforms the normal distribution into a more complicated P^X.

4.1 Transport maps

Before going on with the specific theory of normalizing flows, we first elaborate on the properties of mappings between two probability spaces $(\mathcal{Z}, \mathcal{Z}_{\mathcal{X}}, P^Z)$ and $(\mathcal{X}, \mathcal{A}_{\mathcal{X}}, P^X)$. In this general setting, normalizing flows are related to transportation theory. Specifically, a measurable function $T : \mathcal{Z} \to \mathcal{X}$ such that $T_*P^Z = P^X$ is called a *transport map* [83]. In general, an injective transport map is not guaranteed to exist and is not unique P^Z-almost surely. For instance, consider a transport map from the normal distribution $\mathcal{N}(0,1)$ to the Dirac measure δ on $\mathbb{R}$. Then $T = 0$ almost everywhere, which is not an injective function. Furthermore, if P^Z is a symmetric distribution on $\mathbb{R}$, then the invertible functions T and $T'(x) := T(-x)$ are different transport maps.

Under some assumptions, a transport map does exist. For instance, a bijective transport map between two Polish spaces (complete, separable, and metrizable) with the Borel σ-algebra and with no atoms (points with nonzero probability) does exist [83, Chapter 1]. Another case of interest is when, when $\mathcal{Z} = \mathcal{X} = \mathbb{R}^d$ and P^X and P^Z are absolutely continuous with respect to the Lebesgue measure. Indeed, with this setup, an invertible transport map can be explicitly constructed [41, 68]. This map is named the Knothe-Rosenblatt transport and guarantees that the Jacobian matrix DT is triangular.

For our applications, we need to be able to calculate the probability density $p_{T(Z)}$ of T_*P^Z, where P^Z is a probability with density p_X and $\mathcal{Z} = \mathcal{X} = \mathbb{R}^d$. When T is injective and differentiable almost everywhere then by [83, Theorem 11.1] a formula for $p_T(Z)$ is given by the Jacobian equation

$$p_{T(Z)}(x) = p_Z(T^{-1}(x))\mathcal{J}_{T^{-1}}(x), \tag{4.1}$$

where

$$\mathcal{J}_{T^{-1}}(x) = \left|\det\left(DT^{-1}(x)\right)\right| = \lim_{r\to 0}\frac{\lambda[T(B_r(x))]}{\lambda[(B_r(x))]}$$

is the Jacobian determinant of T^{-1} at x, with $B_r(x)$ and λ denoting a ball of radius r and the Lebesgue measure respectively.

Finally, consider the case in which T is a Lipschitz continuous flow $T = \phi_t$ of a time-dependent vector field

$$\xi : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d.$$

Then the probability density p_t of T_*P^Z is the time-t solution to the mass conservation equation

$$\frac{\partial}{\partial t}p + \nabla \cdot (p\xi) = 0, \tag{4.2}$$

with boundary condition $p_0 = p_Z$, the probability density of P^Z[83].

4.2 Minimizing the Kullback-Leibler divergence

One strategy to find a transformation T is to select a function T that minimizes the cost L, where L is the difference between the probability distributions P^X and T_*P^Z.

$$L(T) = D(P^X, T_*P^Z)$$

Although there are many alternative methods to measure the difference D [61, Section 2.3.4], a usual choice for such a function is the Kullback-Leibler divergence [44].

Definition 4.2.1 (Kullback-Leibler divergence)**.** *For two probability measures P and Q on a measure space $(\Omega, \mathcal{A})$, P absolutely continuous with respect to Q ($P \ll Q$), the Kullbak-Leibler divergence is given by*

$$D_{KL}(P||Q) = \mathbb{E}_P \left[\log \frac{dP}{dQ} \right],$$

where $\frac{dP}{dQ}$ denotes the Radon-Nikodym derivative of P with respect to Q. If P is not absolutely continuous with respect to Q, then $D_{KL}(P||Q) := \infty$.

Remark. *KL-divergence is sometimes defined differently for discrete and continuous probability distributions. However, by [30, Lemma 5.2.3] we get a unified definition for all feasible probability distributions. The definition we use is also the one given by the authors Kullback and Leibler.*

The Radon-Nikodym derivative of P with respect to Q is the Q-almost surely unique nonnegative random variable such that

$$\mathbb{E}_Q \left[\mathbb{1}_A \frac{dP}{dQ} \right] = P(A) \;\forall\; A \in \mathcal{A},$$

for two σ-finite measures P and Q, $P \ll Q$ [69, Theorem 5.4]. Moreover, if $P \ll Q \ll \mu$ for a σ-finite measure μ, then

$$\frac{dP}{d\mu} = \frac{dP}{dQ}\frac{dQ}{d\mu}.$$

Furthermore, since both the Lebesgue measure on $\mathbb{R}^d$ and the counting measure are σ-finite, then using the chain rule above we have

$$\frac{dP}{dQ} = \frac{p_P}{p_Q},$$

where p_P and p_Q are the probability densities of P and Q respectively.

In the case when P is not absolutely continuous with respect to Q then there is a set $A \in \mathcal{A}$ such $P(A) > 0$ and $Q(A) = 0$. Therefore the Radon-Nikodym derivative is not defined but

$$\log \frac{P(A)}{Q(A)} = \infty.$$

Therefore we define $D_{KL}(P||Q) := \infty$ in this case.

KL-divergence has some interesting properties. Firstly, it is always non-negative. To see this we use Jensen's inequality:

$$\mathbb{E}_P\left[\log\frac{dP}{dQ}\right]=\mathbb{E}_P\left[-\log\frac{dQ}{dP}\right]\geq-\log\mathbb{E}_P\left[\frac{dQ}{dP}\right]=0.$$

Moreover, equality follows only when $P=Q\quad P-$almost surely. KL divergence is not symmetric and thus, not a distance function. The asymmetry leads to two ways to measure the discrepancy between P^X and T_*P^Z. The first one is *forward* KL-divergence

$$D_{\mathrm{KL}}(P^X||\,T_*P^Z)=-\mathbb{E}\left[\log p_{T(Z)}(X)\right]-H(P^X), \tag{4.3}$$

where $H(P^X)=\mathbb{E}\left[-\log p_X(X)\right]$ is the *constant* entropy of P^X. Forward KL-divergence can be utilized in the situation where we do not know the distribution P^X, but we can draw samples from it. Therefore, if we have samples $\{x_i\}_{i=1}^m=\mathcal{D}$ from the probability distribution P^X, then a Monte Carlo estimate for the expectation for forward KL-divergence is

$$D_{\mathrm{KL}}(P^X||\,T_*P^Z)\approx-\sum_{i=1}^m\log\left[p_{T(Z)}(x_i)\right]+H. \tag{4.4}$$

Minimizing this expression is also equivalent to maximizing the log-likelihood of the samples $\mathcal{D}$. The relation to log-likelihood motivates the use of forward KL-divergence as a cost function for the optimization problem we will introduce in the next section.

On the other hand, if we cannot sample from P^X, but we can evaluate the probability mass function p_X, then we can still estimate the *reverse* KL-divergence.

$$D_{\mathrm{KL}}(T_*P^Z||\,P^X)=\mathbb{E}\left[\log p_{T(Z)}(T(Z))-\log p_X(T(Z))\right].$$

An estimate is found by generating a sample $\{z_i\}_{i=1}^m$ from P^Z. Then a Monte Carlo estimate for reverse KL-divergence is

$$D_{\mathrm{KL}}(T_*P^Z||\,P^X)\approx\sum_{i=1}^m\left[\log p_{T(Z)}(T(z_i))-\log p_X(T(z_i))\right].$$

4.3 Discrete time normalizing flows

Normalizing flows in discrete time are models where the transformation T is composed of N simpler transformations

$$T=T_N\circ\ldots\circ T_2\circ T_1. \tag{4.5}$$

Then for each discrete time k the a update is defined as

$$z_k=T_k(z_{k-1}).$$

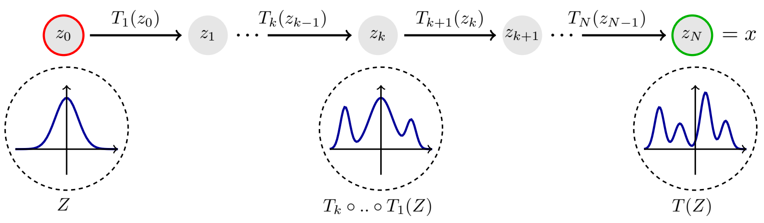


Figure 4.1: Example of how T transforms the random variable Z. Figure inspired by [87]

Thus, as shown in Figure 4.1, the random variable Z is iteratively transformed into X.

Most applications of normalizing flows adds additional constraints on T. Firstly T must be invertible and have an inverse that is efficient to compute. Secondly, T must allow for efficient computation of the Jacobian determinant. To be precise, T is often supposed to be a diffeomorphism.. Furthermore, if T is composed of constituent transformations $\{T_k\}_{k=1}^N$ as in Equation (4.5) and $T_k \in \text{Diff}(\mathbb{R}^n)$ for all $1 \leq i \leq N$. Then we can compute the logarithm of the Jacobian determinant of T efficient with the formula

$$\log \mathcal{J}_{T^{-1}}(x) = -\sum_{k=1}^{N} \log \mathcal{J}_{T_k}(z_{k-1}), \tag{4.6}$$

where $z_k = T_{k+1}^{-1} \circ T_{k+2}^{-1} \circ \cdots \circ T_N^{-1}(x)$ such that $z_N = x \in \mathbb{R}^n$.

We now find a transformation T that minimizes KL-divergence. One common optimization strategy is select T among a set of functions $\{T_\theta\}_{\theta \in \Theta}$ parametrized by a set Θ to which we can apply an optimization algorithm. Moreover, we do not know P^X but we have samples $\{x^i\}_{i=1}^M$ drawn from it. Thus by equations (4.1) and (4.4), we must minimize the cost function

$$\mathcal{L} : \Theta \to \mathbb{R}, \quad \mathcal{L}(\theta) = -\sum_{i=1}^{m} \left[\log p_Z(T_\theta^{-1}(x^i)) + \log \mathcal{J}_{T_\theta^{-1}}(x^i) \right], \tag{4.7}$$

Moreover, using the relation in Equation (4.6), $\mathcal{L}$ can be computed as described in Algorithm 3.

Remark. *Note that the algorithm above computes the Jacobian of the inverse flow T. It is possible to compute D_{KL} using the Jacobian of T^{-1} since* $\log \left|\det J_T(T^{-1}(x))\right| = -\log \left|\det J_{T^{-1}}(x)\right|$

4.3.1 Invertible residual neural networks

There are a lot of different models that have been used to construct the map T [42, 61]. Most of these models restrict the structure of the Jacobian matrix. For instance, autoregressive flows [60] restrict the Jacobian to be triangular.

Algorithm 3 Forward propagation for normalizing flows

```
Require: Samples $\{x^i\}_{i=1}^M$ drawn according to $P^X$
  Initialize $D_{\mathrm{KL}} = 0$
  for $i \leftarrow 1$ to $M$ do
      $z \leftarrow x^i$
      for $k \leftarrow N$ to $1$ do
          $D_{\mathrm{KL}} \leftarrow D_{\mathrm{KL}} - \log \mathcal{J}_{T_k^{-1}}(z)$
          $z \leftarrow T_k^{-1}(z)$
      end for
      $D_{\mathrm{KL}} \leftarrow D_{\mathrm{KL}} - \log p_Z(z)$
  end for
```

However, this restriction also limits the expressiveness of the function. One of the models we consider in this text will be invertible residual neural networks.

Invertible residual networks [7] are invertible functions composed of layers

$$T_i^{-1} = \mathrm{id} + g_i,$$

where g is a network block with a forms similar to

$$g_i = \sigma \circ W_i^L \circ \cdots \circ \sigma \circ W_i^1$$

for a sigmoidal function σ and linear transformations W_i^j. Next, if g_i is a contraction, then by the Banach fixed-point theorem [31] T_i^{-1} is injective with an inverse that can be calculated by fixed point iteration. Therefore, if σ and W_i^j are contractive, then the mapping T^{-1} is injective and has an inverse that is computable by fixed point iteration. Therefore, we select σ to be a contractive function (e.g. ReLU, tanh), and scale each transformation W_i^j using an estimate of its spectral norm $\hat{\sigma}_i^j$

$$\hat{W}_i^j = \begin{cases} W_i^j \ / \ L\hat{\sigma}_i^j & \text{if} \quad L < \hat{\sigma}_i^j \\ W_i^j & \text{else} \end{cases},$$

with $L < 1$. In particular the spectral norm of a matrix W can be computed by power iteration [27] on W^*W. However, as noted by [28], since power iteration calculates an underestimate of the spectral norm $\hat{\sigma} \leq \|W\|_2$, we cannot guarantee that W_i^j is a contraction.

We also need to compute the Jacobian determinant of each layer T_i. Since calculating the determinant has time complexity $\mathcal{O}(d^3)$. A direct method for calculating the Jacobian determinant is unpractical for high-dimensional data. Chen et al.[16] instead uses the fact that g_i is a contraction to derive the formula

$$\mathcal{J}_{T_i^{-1}}(x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} \operatorname{tr}\big[(Dg_i(x))^k\big]}{k}. \tag{4.8}$$

However, this formula still requires $\mathcal{O}(kd^3)$ operations, but by using Hutchinson's trace estimator [35], we estimate each trace using $\mathcal{O}(d^2)$ computations. Hutchinson's trace estimator uses a random variable ϵ satisfying $\mathbb{E}[\epsilon] = 0$ and $\text{Cov}[\epsilon] = I_d$. Then for a matrix $A \in \mathbb{R}^{d\times d}$

$$\text{tr}\{A\} = \mathbb{E}\left[\epsilon^T A\epsilon\right]. \tag{4.9}$$

Moreover, if the sum (4.8) is truncated at a fixed at a fixed term K, then the estimate for $\mathcal{J}_{T_i^{-1}}(x)$ is biased. Therefore, [16] suggests using an estimate the infinite unbiassedly in finite time. Let N be a random variable with support on the positive integers with probability distribution P^N. Then by [16, Theorem]

$$\mathcal{J}_{T_i^{-1}}(x) = \mathbb{E}\left[\sum_{k=1}^{N} \frac{j_k}{P^N(\{n \geq k\})}\right],$$

where j_k is the k-th term of the sum (4.8). Thus the is unbiassedly computable by selecting the number of terms to include through a geometric distribution, and weighing each term appropriately.

4.4 Continuous normalizing flows

A continuous normalizing flow is a model where the transport map T is defined by the flow ϕ_t of a continuous-time dynamical system. More specifically, we define a ODE

$$\begin{aligned} \dot{z}(t) &= f(z(t), t) \\ z(t_0) &= z_0. \end{aligned} \tag{4.10}$$

Then using the ODE we define T as the flow from time t_0 to t_1

$$T(z_0) = \phi_{t_1}(z_0).$$

The ODE formulation provides us with the properties we need to define a normalizing flow. Firstly, if f_θ is continuous and Lipschitz in its first argument, then by Picard-Lindelöf theorem [3] then, the solution $z(t)$ exists and is unique. In addition, the flow ϕ_t of the system is bi-Lipschitz[1]. Since $x = T(z_0)$ is unique, we can compute both T and T^{-1} using the relation

$$x = z_0 + \int_{t_0}^{t_1} f(z(t), t)\, dt.$$

Futhermore, since T is bi-Lipschitz, the Jacobian equation (4.1) determined the probability density $p_{T(Z)}$. In this case there are two strategies for computing $p_{T(Z)}$. Firstly, we could use [15, Theorem 1] to compute the time

[1]A bi-Lipschitz function is an Lipschitz function with an inverse that is also Lipschitz.

derivative of Jacobian at each time t. However, as Chen et al.[15] also notes, the probability $p(z(t),t)$ of $z(t) = \phi_t(z_0)$ is given by a the mass conservation equation (4.2) with vector field $\xi = f$. The characteristic lines of this PDE are given by

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}p(z(t),t) &= \frac{\partial}{\partial z}p(z(t),t)\dot{z}(t) + \frac{\partial}{\partial t}p(z(t),t)\\ &= \frac{\partial}{\partial z}p(z(t),t)f(z(t),t) - \nabla \cdot [p(z(t),z)f(z(t),t)]\\ &= -p(z(t),t)\nabla \cdot f(z(t),t).\end{aligned}$$

Thus we are able to compute the probability $p(z(t_1),t_1) = p_{T(Z)}(z_0)$ by integrating along this characteristic. For our purpose we are interested in the log-probability, which by the the differentiation formula for the logarithm is given by

$$\frac{\mathrm{d}}{\mathrm{d}t}\log p(z(t),t) = -\frac{p(z(t),t)}{p(z(t),t)}\nabla \cdot f_\theta(z(t),t) = -\operatorname{tr}\left(Df(z(t),t)\right). \tag{4.11}$$

Also, as suggested by Grathwohl et al.[29], we can also apply the Hutchinson's trace estimator (4.9) when dealing with high-dimensional data. Finally, the probability density of P^Z and T_*P^X is given by integrating the equations (4.10) and (4.11)

$$\begin{bmatrix} T(z_0) \\ \log p_{T(Z)}(z_0)\end{bmatrix} = \begin{bmatrix} z_0 \\ \log p_Z(z_0)\end{bmatrix} + \int_{t_0}^{t_1} \begin{bmatrix} f(z(t),t) \\ -\operatorname{tr}\left(Df(z(t),t)\right)\end{bmatrix} dt, \tag{4.12}$$

which can be computed using a normal ODE solver.

The continuous nature of our system also allows us to optimize a cost function using the adjoint equation. If we have a cost $\mathcal{L}$ that is dependent on $z(t_1)$, and a vector field f_θ is parametrized by $\theta \in \Theta$. Then, as remarked by Chen et al.[15], the gradient of the cost function can be computed by integrating backwards via the adjoint state ODE

$$\begin{aligned}\dot{a}(t) &= -a(t)^T \frac{\partial}{\partial z} f_\theta(z(t),t)\\ a(t_1) &= \frac{\mathrm{d}}{\mathrm{d}z(t_1)}\mathcal{L}.\end{aligned}$$

Then the gradient of $\mathcal{L}$ can be computed by the integral

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\mathcal{L} = \int_{t_0}^{t_1} a(t)^T \frac{\partial}{\partial \theta} f_\theta(z(t),t)\, dt$$

Moreover, when optimizing the mapping T in the normalizing flow model we have to include both terms from Equation (4.12) for the parametrized vector field f_θ. Then we can use the loss function given by KL-divergence (4.4). Thus, we can calculate probabilities and optimize the model by integrating ODEs.

4.5 Sampling and interpolation

Having an approximate probability distribution on the space $\mathcal{X}$ allows us to sample from the probability space by generating samples. By Equation (4.1), if we choose a sample in $z \in \mathcal{Z}$ with a high probability, and if the Jacobian determinant $\mathcal{J}_T(z)$ is large, then the probability of our sample $T(z)$ will also have a high probability. However, the Jacobian need not be small, in a high dimensional setting the a small volume increase in each direction of the space can lead to a very large Jacobian determinant $\mathcal{J}_T(z)$ and thus a very small value of $p_{T(Z)}(T(z))$.

Using normalizing flows we are also able to interpolate in the *latent* space $\mathcal{Z}$ rather than in the *feature* space $\mathcal{X}$ as seen in [40]. For two points $x_1, x_2 \in \mathcal{X}$ the interpolating path given by linear interpolation in the latent space is given by

$$\nu(t) = T(tT^{-1}(x_1) + (1-t)T^{-1}(x_2)) \quad t \in [0,1]$$

Moreover, since the normal distribution is a concave function, then if the Jacobian determinant is constant along the path, then

$$p_{T(Z)}(\nu(t)) \geq \min\{p_{T(Z)}(x_1), p_{Z(T)}(x_2)\} \quad \forall t \in [0,1].$$

Thus, if we use latent space interpolation to interpolate two points with a high probability density, then we would expect each point along the path ν to have a high probability density as well.

Using this interpolation technique enables the generation of new data based on existing data points. Additionally, using multiple samples from the feature we can construct interpolation surfaces of arbitrary. One example of this interpolation is shown in [19], where they generate new pictures based on a nonlinear surface in the latent space interpolating old pictures.

4.6 Clustering with normalizing flows

To cluster data we follow the approach by [1]. We define a hidden discrete random variable C taking values in $\mathcal{K}$ and with a probability distribution P^Csuch that law$(C) = P^Z$. Then for each of the sampled data we for each value $k \in C$ we define a normalizing flow $T_k : \mathcal{X} \to \mathcal{X}$ as before. Using the set of normalizing flows we define a new mapping

$$\mathcal{T} : \mathcal{Z} \times \mathcal{K} \to \mathcal{X} \times \mathcal{K}, \quad \mathcal{T}(z,k) = T_k(z) \times k.$$

Thus, we can define a joint probability distribution on $\mathcal{X} \times \mathcal{K}$ by using the pushforward probability $\mathcal{T}_*(P^Z \otimes P^C)$, where $P^Z \otimes P^C$ is the probability distribution on $\mathcal{Z} \times \mathcal{K}$ so that Z and C are independent [36, Theorem 10.3]. Moreover, each map T_k defines the conditional probability density $p_{T_k(Z)}(X)$

of X given $C = k$. Using this conditional probability density we get an expression for the proabily density of $\mathcal{T}_*(P^Z \otimes P^C)$ given by

$$p_{\mathcal{T}(Z,C)}(x,k) = p_{T_k(Z)}(x)p_C(k),$$

where p_C is the probability density of P^C.

4.6.1 Expectation maximization

In a similar approach to [1] we will utilize the expectation maximization (EM) algorithm [18] to determine each of the maps $\{T_k\}_{k\in\mathcal{K}}$. Agnelli et al. define a flow that minimizes the KL-divergence, then taking EM steps in the flow direction. However, we will apply the EM algorithm directly by minimizing the KL-divergence between the true probability $P^{(X,C)}$ and $\mathcal{T}_*(P^Z \otimes P^C)$. We start by using the definition of conditional expectation [36, Definition 23.4] and Equation (4.6)

$$\begin{aligned} D_{\mathrm{KL}}(P^{(X,C)} \| \, \mathcal{T}_*(P^Z \otimes P^C)) &= -\mathbb{E}\left[\mathbb{E}\left[\log p_{\mathcal{T}(Z,C)}(X,C) \middle| X\right]\right] - H(P^{(X,C)}) \\ &= -\mathbb{E}\left[\mathbb{E}\left[\log p_{T_C(Z)}(X) \middle| X\right]\right] \\ &\quad - \mathbb{E}\left[\mathbb{E}\left[\log p_C(C) \middle| X\right]\right] - H(P^{(X,C)}), \end{aligned}$$

to get an expression for the KL-divergence where $H(P^{(X,C)})$ is the entropy of $P^{(X,C)}$. We also note that $-\mathbb{E}\left[\mathbb{E}\left[\log p_C(C) \middle| X\right]\right] - H(P^{(X,C)})$ does not depend on $\mathcal{T}$.

Remark. *For a discrete random random variable C and a measurable function $f : \mathcal{X} \times \mathcal{K} \to \mathbb{R}$, by [36, Definition 23.1] conditional expectation is given by*

$$\mathbb{E}\left[f(X,C) \middle| X\right] = \sum_{k\in\mathcal{K}} p_{C|X}(k,X) f(X,k),$$

where $p_{C|X}$ is the conditional probability density of C given X. In addition, if we have samples $\{x^i\}_{i=0}^N$ from the probability distribution P^X, then a Monte Carlo estimate of the expectation can be used as in Equation (4.4).

Since we cannot observe C, we are not able to minimize the KL-divergence directly. Instead, we follow the *expectation step* in the EM procedure, and make a guess $p_{C|X} \approx \hat{p}_{C|X}$ using Bayes' theorem [36, Theorem 3.5]

$$\hat{p}_{C|X}(c|x) = \frac{p_{T_k}(x)p_C(c)}{\sum_{k\in\mathcal{K}} p_{T_k}(x)p_C(k)}. \tag{4.13}$$

Then in the *maximization step*, we find the map $\mathcal{T}$ that maximize the negative KL-divergence using the estimate $\hat{p}_{C|X}$. Since only the first term in the expression for KL-divergence depends on $\{T_k\}_{k\in\mathcal{K}}$ we only need minimize this first term. Thus, while using the constant approximation $p_{C|X} \approx \hat{p}_{C|X}$, the maximization step involved finding $\{T_k\}_{k\in\mathcal{K}}$ that maximizes

$$\mathbb{E}\left[\sum_{k\in\mathcal{K}} \hat{p}_{C|X}(k|X) \log p_{T_k(Z)}(X)\right]. \tag{4.14}$$

The expectation and maximization step is then repeated until convergence.

Two modifications can be made to the EM algorithm. Firstly by finding $\{T_k\}_{k\in\mathcal{K}}$ maximizing (4.14) can lead to a very difficult optimization problem. Thus, in each maximization step, we perform one step of an optimization algorithm instead. Secondly, as noted by [1], for each data point $x^i \in \mathcal{X}$ we can have different prior distributions P_i^C. Thus if we have labels for some of the data points we can use these when calculating $\hat{p}_{C|X}$. For unlabeled data points, we are still able to choose a guess P_i^C. This leads to a semi-supervised learning algorithm where we can utilize both labeled and unlabeled training points.

4.6.2 KL-divergence of the marginal distribution

The probability distribution $\mathcal{T}_*(P^Z \otimes P^C)$ also defines a marginal probability on the space $\mathcal{X}$ with density

$$\hat{p}_X(x) = \sum_{k\in\mathcal{K}} p_{T_k}(x|k) p_C(k).$$

Thus we would also like the EM algorithm to minimize the KL-divergence $D_{\mathrm{KL}}(p_X || \hat{p}_X)$. Using a proof adapted from [25, Section 4.2.1] we show that the KL-divergence of the marginal distribution does decrease with each step of the EM algorithm.

Proposition 4.6.1. *Let $\hat{p}$ and $\hat{p}'$ denote two successive probability densities generated by the EM algorithm on the space $\mathcal{X} \times \mathcal{K}$. Moreover, let $\hat{p}_X$ be the marginal probability of $\hat{p}$ for a step of the EM algorithm. Then each step of the EM algorithm decreasing* (4.14)*, also decreases $D_{KL}(p_X || \hat{p}_X)$, .*

Proof. Let $\hat{p}_{C|X}, \hat{p}'_{C|X}$ denote the conditional probability densities of $\hat{p}$ and $\hat{p}'$ respectively. Then, since the maximization step maximizes (4.14) we have

$$\mathbb{E}\left[\mathbb{E}_{\hat{p}}\left[\log \hat{p}'_{X|C}(X|C) - \log \hat{p}_{X|C}(X|C) \middle| X\right]\right] \geq 0.$$

Moreover by appling the logarithm to Bayes' theorem we have the relation

$$\begin{aligned}\log \hat{p}_X(x) - \log \hat{p}'_X(x) &= \log \hat{p}_{X|C}(x|c) - \log \hat{p}'_{X|C}(x|c)\\ &\quad - \log \hat{p}_{C|X}(c|x) + \log \hat{p}'_{C|X}(c|x).\end{aligned}$$

Now using the two relations above and Jensen inequality for conditional expectation [36, Theorem 23.9] we calculate

$$\begin{aligned}
D_{\mathrm{KL}}(p_X||\,\hat{p}'_X) - D_{\mathrm{KL}}(p_X||\,\hat{p}_X) &= \mathbb{E}\left[\log \hat{p}_X(X) - \log \hat{p}'_X(X)\right] \\
&\leq \mathbb{E}\left[\mathbb{E}_{\hat{p}}\left[-\log \hat{p}_{C|X}(C|X) + \log \hat{p}'_{C|X}(C|X)\middle| X\right]\right] \\
&\leq \mathbb{E}\left[\log \mathbb{E}_{\hat{p}}\left[\frac{\hat{p}'_{C|X}(C|X)}{\hat{p}_{C|X}(C|X)}\middle| X\right]\right] \\
&= \mathbb{E}\left[\log \mathbb{E}_{\hat{p}'}\left[1|X\right]\right] = 0
\end{aligned}$$

□

4.6.3 Clustering performance and variable selection

When the dimensions of the feature space $\mathcal{X}$ gets large, clustering with normalizing flows becomes both computationally and conceptually harder. To remedy the problem in high dimensions [1] proposes a method for variable selection. They suggest clustering each dimension of the problem separately, then choosing the dimensions that enable the "best" clustering.

The performance of a clustering given by the conditional probably density $\hat{p}_{C|X}$ can also be related to KL-divergence. In this approach we formulate a new probably density $\hat{p}$ on $\mathcal{X} \times \mathcal{K}$ by

$$\hat{p}(x,k) = \hat{p}_{C|X}(k|x)p_X(x),$$

where p_X is the true probability density of $\mathcal{X}$. Then the KL-divergence between the estimated probability density $\hat{p}$ and p is given by

$$D_{\mathrm{KL}}(p \mid\mid \hat{p}) = H(p, \hat{p}|X) - H(p|X), \tag{4.15}$$

where

$$H(p, \hat{p}|X) = -\mathbb{E}\left[\mathbb{E}\left[\log \hat{p}_{C|X}(C|X)\middle|X\right]\right], \tag{4.16}$$

is the conditional cross entropy of p and $\hat{p}$ given X, and

$$H(p|X) = -\mathbb{E}\left[\mathbb{E}\left[\log p_{C|X}(C|X)\middle|X\right]\right],$$

is the conditional entropy of p given X.

Thus, if the true conditional probability density $p_{C|X}$ is available, then, a measure of how well p supports a clustering $\hat{p}_{C|X}$ is given by the negative conditional cross-entropy $-H(p, \hat{p}|X)$. The reason for using this quantity is that the conditional entropy of p given X $H(p|X)$ does not depend on $\hat{p}_{X|C}$. Thus, minimizing KL-divergence with respect to $\hat{p}_{C|X}$ is equivalent to minimizing $H(p, \hat{p}|X)$. On the other hand, if $p_{C|X}$ is not available, then we use the maximum value of $-H(p, \hat{p}|X)$ for all probability densities p. By Equation (4.15) $-H(p, \hat{p}|X)$ achieves a maximum when $p = \hat{p}$. Thus, when $p_{C|X}$ is not available we estimate $-H(p, \hat{p}|X)$ by the negative conditional entropy of $\hat{p}$ given X $H(\hat{p}|X)$.

4.7 Experiments

In this section, we will interpolate and cluster test examples and real motion capture data from the *CMU Graphics Lab Motion Capture Database*. As seen in Section 2.4, human motion on the interval I can be seen as a curve $I \to SO(3)^d$. In this section, we will not use the SRV form of the curve. Instead, for each of the d joints in the human body we parametrize the rotation matrix $M \in SO(3)$ with Euler angles ϕ, θ and ξ [26, Section 4.4]. In addition, we will disregard the position and rotation of the body relative to the room. Then accounting for the fact that not all joints have three degrees of freedom, the total number of angles in each frame of human motion is 44. Thus for each captured frame of human motion, we have to store a vector in $\mathbb{R}^{44}$.

In these experiments, all datasets were standardized before interpolation or clustering was performed. Moreover for all experiments using continuous normalizing flows we used a neural network [17] as an autonomous vector field, and RK5(4) [20] as the numerical integration Algorithm. For more information about implementation and parameters, see Appendix A.2.

4.7.1 Interpolation

In these experiments, we trained normalizing flows to approximate the distribution of three different datasets. Then we choose two samples and performed latent and feature space interpolation. Then, the probability densities of each point in the resulting paths were compared to each other.

Moons

The dataset used in this experiment is the `moons` dataset from [63] with noise parameter of 0.05. We used a model with four continuous normalizing flow blocks with time derivatives given by neural networks. The resulting paths and probabilities are shown in 4.2

Human motion frames

In this experiment, we used a normalizing flow with invertible residual networks. We picked 65 walking motions and 44 running motions with a total of 31858 frames. Feature and latent space interpolation was then done between one walking frame and one running frame. The result of the interpolation can be seen in Figure 4.3

Human motion

In this experiment, we used a normalizing flow with invertible residual networks with two dimensional convolution layers. We picked 65 walking mo-

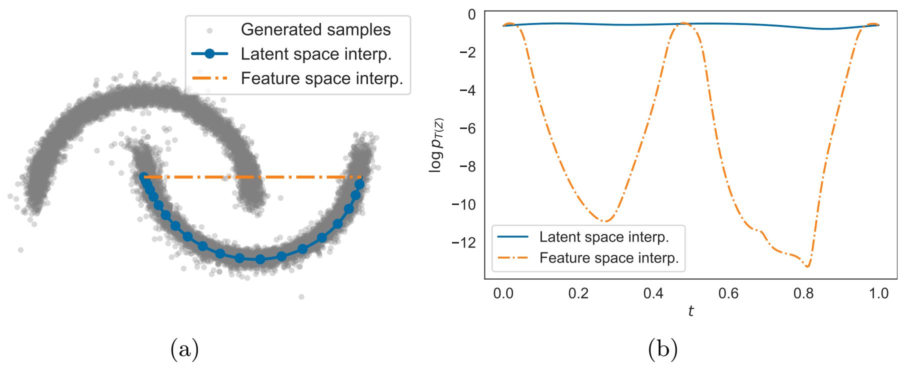


Figure 4.2: Figures shows the interpolated paths (a) and probability density at each point of the paths as given by $p_{T(Z)}$ (b). The paths where generated by feature space interpolation and latent space interpolation between the two endpoints.

tions with 120 frames each. Thus each data point x fed to the network was a matrix $x \in \mathbb{R}^{120\times 44}$. Feature and latent space interpolation was then done between one two walking frames. The result of the interpolation can be seen in Figure 4.4

4.7.2 Clustering

In these experiments, we test the clustering algorithm on four different datasets. In the cases where unsupervised clustering is not successful, we also attempt semi-supervised clustering with some labeled points.

Moons and circles

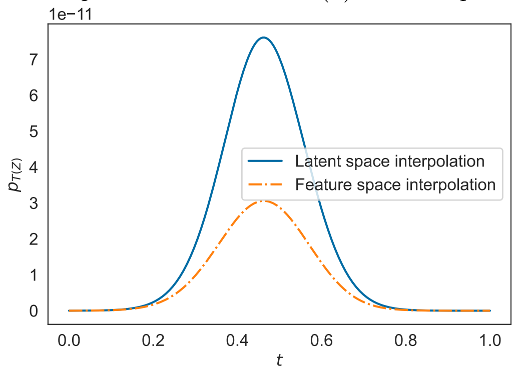
The first test dataset we attempt to cluster is the `moons` dataset with a noise parameter of 0.05. The result for this dataset can be seen in Figure 4.5. The second dataset we attempt to cluster is the `circles` dataset from [63] with a noise parameter of 0.01. The result of both supervised and semi-supervised clustering for the `circles` dataset is shown in Figure 4.6

Human motion frames

In this experiment, we selected 65 walking motions and 44 running with a total of 31858 frames. We attempted to cluster all of these frames into two classes, but we did not get good results using all 44 Euler angles simultaneously. Instead, we first performed variable selection by choosing the angles that allow for best data clustering, in the same manner as Agnelli et al.[1] suggests. We clustered each angle separately for each of the 44 angles, producing 44 conditional probability densities $\hat{p}_{C|X}$. We then selected the five angles that produced the negative highest conditional entropy of $\hat{p}$ given X. Using these five angles we clustered the walking and running frames into

(a) Latent space interpolation

(b) Feature space interpolation



(c)

Figure 4.3: Linear interpolation in feature and latent space between one walking frame and one running frame. The probability density of each path with respect to the probability $p_{T(Z)}$ is shown in (c). The video showing each interpolation can bee seen by scanning the QR-codes, (a) and (b).

two classes as shown in Table 4.1. For two of the angles, we also show the marginal probability densities of p_{T_0} and p_{T_1} in Figure 4.7.

We also investigate whether providing labels to 10% of the training data improves the resulting classification. Using the same five Euler angles as before we performed semi-supervised classification, with the result shown in Table 4.2.

Class	1	2
Walk frames	5259 (21%)	19512 (79%)
Run frames	6883 (97%)	204 (3%)

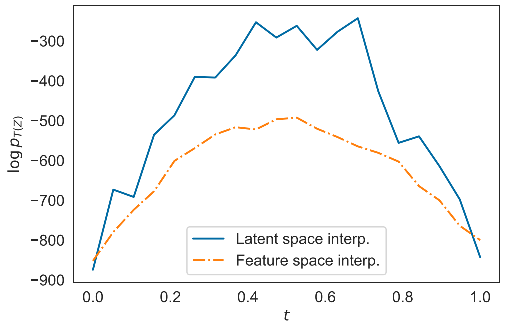
Table 4.1: Table shows the classification of running and walking frames for unsupervised clustering. The number of frames in each class is shown, separately for frames from walking movements and running movements.

Human motions

In this final experiment, we attempt to cluster motion into two classes. We consider one second of 65 running motions and 44 running motions with 10

(a) Latent space interpolation

(b) Feature space interpolation



(c) Probability densities

Figure 4.4: Linear interpolation in feature and latent space between two walking motions. Since each motion is a sequence of frames, the interpolated path is shown as a sequence of movies that can bee seen by scanning the QR codes (a) and (b). The probability density of each path with respect to the probability $p_{T(Z)}$ is shown in (c).

frames per second. We will also only look at the Euler angles chosen by the previous experiment clustering frames of human motion. Thus, each data point x^i is a matrix $x^i \in \mathbb{R}^{10\times5}$. Moreover, to reduce the number of parameters in the model we will use a CNF with the time derivative given by a convolutional neural network. The resulting classification into two classes is shown in Table 4.3.

4.8 Discussion

From the test experiments on the `moons` and `cirlces` datasets, we can see that continuous normalizing flows can learn distributions quite well. Furthermore, linear interpolation in the latent space does produce paths that have a higher probability than feature space interpolation with the probability density $P_T(Z)$. However, the benefits of this interpolation on data from motion capture are not noticeably different from normal feature space interpolation.

As shown in Figure 4.6 clustering with the EM algorithm does not always produce the clusters that humans would expect. However, by using some labeled data we classified each circle in the `circles` dataset correctly.

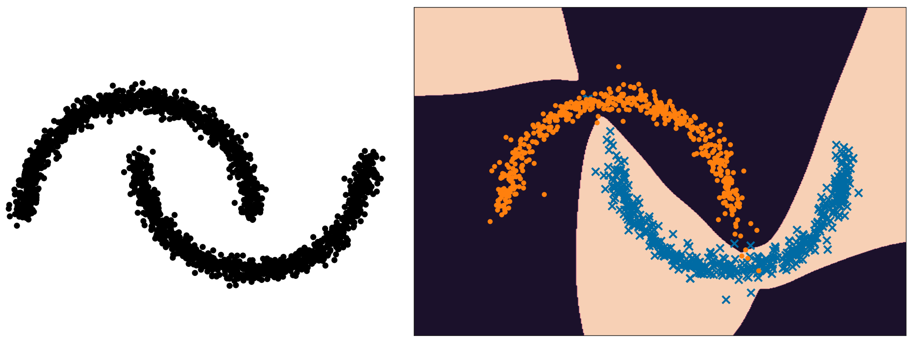


Figure 4.5: Figure shows unsupervised clustering (right) using the unlabeled training points (left) from the `moons` dataset . The background colors shows the decision border for the clustering. The blue cross and orange dots are samples from the two conditional distributions produced by each normalizing flow.

Class	1	2
Walk frames	21259 (86%)	3512 (14 %)
Run frames	288 (4%)	6799 (96 %)

Table 4.2: Table shows the classification of running and walking frames for semi-unsupervised clustering, where 10 % of the training points where labeled. The number of frames in each class is shown, separately for frames from walking movements and running movements.

Moreover, we were also able to cluster walking and running frames somewhat correctly. In this regard, it is worth noting that a lot of frames from the running dataset do look like people walking. Finally, as seen in Table 4.3 we were able to cluster walking and running motions correctly without any labels.

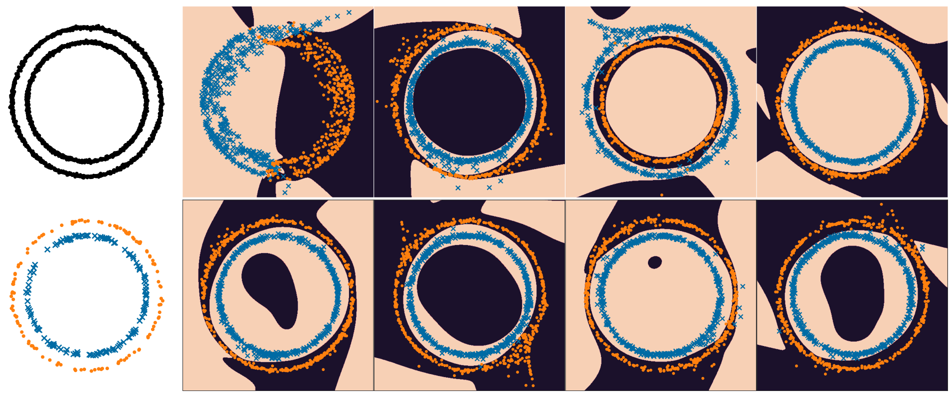


Figure 4.6: Figure shows unsupervised clustering (top) using the unlabeled training points (top left), compared to to semi-supervised clustering (bottom) using both unlabeled and labeled training points (left) from the circles dataset. The background colors shows the decision border for the clustering. The blue cross and orange dots are samples from the two conditional distributions produced by each normalizing flow.

Class	1	2
Walk motions	0	65
Run motion	44	0

Table 4.3: Table shows the classification of running and walking frames for unsupervised clustering. The number of frames in each class is shown, separately for frames from walking movements and running movements.

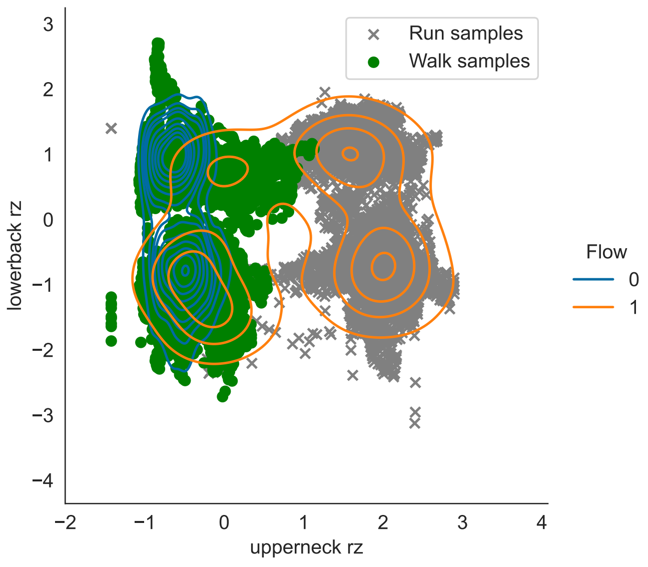


Figure 4.7: Figure shows two of the five Euler angeles use to cluster frames, these two angles are the ones that subjectively showed two clusters the best. The points depicted are the two angles for each training point colored by the class they belong to. Shows the marginal distribution of each of the normalizing flow on each of the two angles. Calculated by sampling from the normaling flows then averaging over each of the angles not displayed. from the, i. e. a Monte Carlo estimate.

Chapter 5

Conclusion

In this thesis, we have discussed two different approaches for comparing curves. In the first approach, an adapted version of deep Q-networks was tested in order to compute an optimal reparametrization. This algorithm did have a linear time complexity with respect to the size of the state space. However, in our tests the algorithm was unreliable, converging to a solution better than the greedy solution in less than half of all our tests. Therefore, even though the alterations we made did make the algorithm more stable, deep Q-networks are not a practical approach for finding optimal reparametrizations.

Normalizing flows were also tested as a method for interpolation and clustering. Using latent space interpolation on human motions and frames of human motions, we were able to create interpolated points with a higher computed probability density than feature space interpolation. Thus we were able to generate new walking motions the model had not seen before. However, when viewing the resulting motions there was no visible improvement from normal feature space interpolation. Furthermore, the proposed clustering algorithm did cluster 78% of walking and running frames into the right cluster. Furthermore, when the clustering method was applied to one second of motion capture data, it was able to cluster all walking motions and running motions into separate clusters. Thus normalizing flows provides an alternative method of clustering motions compared to dynamic programming and SRVT [14]. Moreover, the normalizing flow approach can also cluster new motions without computing the distance to all other motions.

For further work on using reinforcement learning for optimal reparametrization, there are a lot of algorithms that can be tried. Firstly, one question is whether there would be an improvement if an algorithm used a continuous action space instead of a discrete one, for instance, deterministic policy gradient algorithms [72].

For further work on normalizing flows, we also have some suggestions. Firstly, the expressiveness of many normalizing flows, including continuous normalizing flows, is still an open question. Furthermore, it would be interesting if a better latent space interpolation method could be constructed by selecting a path more deliberately than just interpolating in a straight line. For instance, a path that maximized the average probability density of each point would arguably be better, but finding this path is not nessesearly easy. Finally, a more vague question is whether normalizing flows could learn to recognize reparametrizations of the same data. Since humans easily recognize running motions regardless of the frame rate, the idea is not impossible.

Bibliography

[1] Juan P Agnelli et al. "Clustering and classification through normalizing flows in feature space". In: *Multiscale Modeling & Simulation* 8.5 (2010), pp. 1784–1802.

[2] Alexander Johan Alexander. "Reparametrization of curves by neural networks". 2022.

[3] Herbert Amann. *Ordinary Differential Equations: An Introduction to Nonlinear Analysis*. De Gruyter, 2011. ISBN: 9783110853698. DOI: `doi:10.1515/9783110853698`. URL: `https://doi.org/10.1515/9783110853698`.

[4] Martin Bauer, Martins Bruveris, and Peter W. Michor. "Overview of the Geometries of Shape Spaces and Diffeomorphism Groups". In: *Journal of Mathematical Imaging and Vision* 50.1-2 (Jan. 2014), pp. 60–97. ISSN: 1573-7683. DOI: `10.1007/s10851-013-0490-z`. URL: `http://dx.doi.org/10.1007/s10851-013-0490-z`.

[5] Martin Bauer, Markus Eslitzbichler, and Markus Grasmair. "Landmark-guided elastic shape analysis of human character motions". In: *Inverse Problems & Imaging* 11.4 (2017), pp. 601–621.

[6] Martin Bauer et al. "Constructing reparameterization invariant metrics on spaces of plane curves". In: *Differential Geometry and its Applications* 34 (June 2014), pp. 139–165. ISSN: 0926-2245. DOI: `10.1016/j.difgeo.2014.04.008`. URL: `http://dx.doi.org/10.1016/j.difgeo.2014.04.008`.

[7] Jens Behrmann et al. "Invertible Residual Networks". In: (2018). DOI: `10.48550/ARXIV.1811.00995`. URL: `https://arxiv.org/abs/1811.00995`.

[8] Richard Bellman. *Dynamic programming.* eng. Princeton, N.J: Princeton University Press, 1957.

[9] Justin Boyan and Andrew Moore. "Generalization in reinforcement learning: Safely approximating the value function". In: *Advances in neural information processing systems* 7 (1994).

[10] Greg Brockman et al. *OpenAI Gym.* 2016. eprint: `arXiv:1606.01540`.

[11] Martins Bruveris. “Optimal Reparametrizations in the Square Root Velocity Framework”. In: *SIAM Journal on Mathematical Analysis* 48.6 (2016), pp. 4335–4354. DOI: 10.1137/15M1014693. eprint: https://doi.org/10.1137/15M1014693. URL: https://doi.org/10.1137/15M1014693.

[12] Martins Bruveris, Peter w. Michor, and David Mumford. “Geodesic Completeness for Sobolev Metrics on the Space of Immersed Plane Curves”. In: *Forum of Mathematics, Sigma* 2 (2014), e19. DOI: 10.1017/fms.2014.19.

[13] Elena Celledoni, Markus Eslitzbichler, and Alexander Schmeding. “Shape analysis on Lie groups with applications in computer animation”. In: *Journal of Geometric Mechanics* 8.3 (2016), pp. 273–304. ISSN: 1941-4889. DOI: 10.3934/jgm.2016008. URL: http://dx.doi.org/10.3934/jgm.2016008.

[14] Elena Celledoni, Pål Erik Lystad, and Nikolas Tapia. “Signatures in Shape Analysis: An Efficient Approach to Motion Identification”. In: *Geometric Science of Information.* Ed. by Frank Nielsen and Frédéric Barbaresco. Cham: Springer International Publishing, 2019, pp. 21–30. ISBN: 978-3-030-26980-7.

[15] Ricky T. Q. Chen et al. “Neural Ordinary Differential Equations”. In: *Advances in Neural Information Processing Systems.* Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

[16] Ricky T. Q. Chen et al. “Residual Flows for Invertible Generative Modeling”. In: *Advances in Neural Information Processing Systems.* 2019.

[17] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274.

[18] A. P. Dempster, N. M. Laird, and D. B. Rubin. “Maximum Likelihood from Incomplete Data via the EM Algorithm”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38. ISSN: 00359246. URL: http://www.jstor.org/stable/2984875 (visited on 06/26/2022).

[19] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using real nvp”. In: *arXiv preprint arXiv:1605.08803* (2016).

[20] J.R. Dormand and P.J. Prince. "A family of embedded Runge-Kutta formulae". In: *Journal of Computational and Applied Mathematics* 6.1 (1980), pp. 19–26. ISSN: 0377-0427. DOI: `https://doi.org/10.1016/0771-050X(80)90013-3`. URL: `https://www.sciencedirect.com/science/article/pii/0771050X80900133`.

[21] Yonathan Efroni et al. "Beyond the One-Step Greedy Approach in Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning.* Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1387–1396. URL: `https://proceedings.mlr.press/v80/efroni18a.html`.

[22] Markus Eslitzbichler. "Modelling character motions on infinite-dimensional manifolds". In: *The Visual Computer* 31 (2014), pp. 1179–1190.

[23] Vincent François-Lavet et al. "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends in Machine Learning* 11.3-4 (2018), pp. 219–354. ISSN: 1935-8245. DOI: `10.1561/2200000071`. URL: `https://arxiv.org/abs/1811.12560`.

[24] Mevlana C. Gemici, Danilo Rezende, and Shakir Mohamed. *Normalizing Flows on Riemannian Manifolds.* 2016. DOI: `10.48550/ARXIV.1611.02304`. URL: `https://arxiv.org/abs/1611.02304`.

[25] Geof H Givens and Jennifer A Hoeting. *Computational statistics.* Vol. 703. John Wiley & Sons, 2012. DOI: `https://doi.org/10.1002/9781118555552`.

[26] Herbert Goldstein, Charles Poole, and John Safko. *Classical mechanics.* 2002.

[27] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Third. The Johns Hopkins University Press, 1996.

[28] Henry Gouk et al. "Regularisation of neural networks by enforcing lipschitz continuity". In: *Machine Learning* 110.2 (2021), pp. 393–416.

[29] Will Grathwohl et al. "FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models". In: *CoRR* abs/1810.01367 (2018). arXiv: `1810.01367`. URL: `http://arxiv.org/abs/1810.01367`.

[30] Robert M Gray. *Entropy and Information Theory.* ger ; eng. 2. Aufl. New York, NY: Springer Science + Business Media, 2011. ISBN: 1441979697.

[31] Christopher Heil. *Metrics, Norms, Inner Products, and Operator Theory.* Birkhäuser Cham, 2011. ISBN: 978-3-319-65322-8. URL: `https://doi.org/10.1007/978-3-319-65322-8`.

[32] Gustav Eje Henter, Simon Alexanderson, and Jonas Beskow. "MoGlow: Probabilistic and controllable motion synthesis using normalising flows". In: *ACM Transactions on Graphics* 39.4 (2020), 236:1–236:14. DOI: `10.1145/3414685.3417836`.

[33] Emiel Hoogeboom et al. "Integer Discrete Flows and Lossless Compression". In: *CoRR* abs/1905.07376 (2019). arXiv: `1905.07376`. URL: `http://arxiv.org/abs/1905.07376`.

[34] Ronald A Howard. *Dynamic programming and markov processes.* eng. The MIT Press, 1960.

[35] M.F. Hutchinson. "A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines". In: *Communications in Statistics - Simulation and Computation* 19.2 (1990), pp. 433–450. DOI: `10.1080/03610919008812866`. URL: `https://doi.org/10.1080/03610919008812866`.

[36] Jean Jacod and Philip Protter. *Probability Essentials.* eng. Second Edition. Universitext. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2004. ISBN: 3540438718.

[37] Shantanu H. Joshi et al. "Statistical shape analysis of the corpus callosum in Schizophrenia". In: *NeuroImage* 64 (2013), pp. 547–559. ISSN: 1053-8119. DOI: `https://doi.org/10.1016/j.neuroimage.2012.09.024`. URL: `https://www.sciencedirect.com/science/article/pii/S1053811912009342`.

[38] David G. Kendall. "Shape Manifolds, Procrustean Metrics, and Complex Projective Spaces". In: *Bulletin of the London Mathematical Society* 16.2 (1984), pp. 81–121. DOI: `https://doi.org/10.1112/blms/16.2.81`. eprint: `https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/blms/16.2.81`. URL: `https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/blms/16.2.81`.

[39] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[40] Durk P Kingma and Prafulla Dhariwal. "Glow: Generative flow with invertible 1x1 convolutions". In: *Advances in neural information processing systems* 31 (2018).

[41] Herbert Knothe. "Contributions to the theory of convex bodies." In: *Michigan Mathematical Journal* 4.1 (1957), pp. 39 –52. DOI: `10.1307/mmj/1028990175`. URL: `https://doi.org/10.1307/mmj/1028990175`.

[42] I Kobyzev, SJD Prince, and MA Brubaker. "Normalizing Flows: An Introduction and Review of Current Methods, arXiv e-prints". In: *arXiv preprint arXiv:1908.09257* (2019).

[43] P. Kriegl A. Michor. *The Convenient Setting of Global Analysis.* American Mathematical Societ, 1997. URL: `https://doi.org/http://dx.doi.org/10.1090/surv/053`.

[44] S. Kullback and R. A. Leibler. “On Information and Sufficiency”. In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79 –86. DOI: `10.1214/aoms/1177729694`. URL: `https://doi.org/10.1214/aoms/1177729694`.

[45] Manoj Kumar et al. “Videoflow: A conditional flow-based model for stochastic video generation”. In: *arXiv preprint arXiv:1903.01434* (2019).

[46] Sebastian Kurtek and Anuj Srivastava. “Handwritten Text Segmentation Using Elastic Shape Analysis”. In: *2014 22nd International Conference on Pattern Recognition*. 2014, pp. 2501–2506. DOI: `10.1109/ICPR.2014.432`.

[47] Sebastian Kurtek et al. “Statistical analysis of manual segmentations of structures in medical images”. In: *Computer Vision and Image Understanding* 117.9 (2013), pp. 1036–1050. ISSN: 1077-3142. DOI: `https://doi.org/10.1016/j.cviu.2012.11.014`. URL: `https://www.sciencedirect.com/science/article/pii/S1077314213000714`.

[48] CMU Graphics Lab. *CMU Graphics Lab Motion Capture Database*. URL: `http://mocap.cs.cmu.edu/` (visited on 11/24/2021).

[49] Jose Laborde et al. “RNA global alignment in the joint sequence-structure space using elastic shape analysis”. In: *Nucleic Acids Research* 41.11 (Apr. 2013), e114–e114. ISSN: 0305-1048. DOI: `10.1093/nar/gkt187`. eprint: `https://academic.oup.com/nar/article-pdf/41/11/e114/25338218/gkt187.pdf`. URL: `https://doi.org/10.1093/nar/gkt187`.

[50] Hamid Laga et al. “Landmark-free statistical analysis of the shape of plant leaves”. In: *Journal of Theoretical Biology* 363 (2014), pp. 41–52. ISSN: 0022-5193. DOI: `https://doi.org/10.1016/j.jtbi.2014.07.036`. URL: `https://www.sciencedirect.com/science/article/pii/S0022519314004500`.

[51] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2019. arXiv: `1509.02971 [cs.LG]`.

[52] Long-Ji Lin. “Self-improving reactive agents based on reinforcement learning, planning and teaching”. In: *Machine learning* 8.3 (1992), pp. 293–321.

[53] Peter W. Michor and David Mumford. “Riemannian Geometries on Spaces of Plane Curves”. In: *Journal of the European Mathematical Society* 8 (2003), pp. 1–48.

[54] Peter W. Michor and David Mumford. “Vanishing Geodesic Distance on Spaces of Submanifolds and Diffeomorphisms”. In: *Documenta Mathematica* (2004).

[55] Washington Mio, Anuj Srivastava, and Shantanu Joshi. "On Shape of Plane Elastic Curves". In: *International Journal of Computer Vision* 73.3 (2007), pp. 307–324. ISSN: 1573-1405. DOI: `10.1007/s11263-006-9968-0`. URL: `https://doi.org/10.1007/s11263-006-9968-0`.

[56] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533. ISSN: 1476-4687. DOI: `10.1038/nature14236`. URL: `https://doi.org/10.1038/nature14236`.

[57] P.J. Olver. *Equivalence, Invariants, and Symmetry.* Cambridge University Press, 1995. URL: `http://dx.doi.org/10.1090/surv/053`.

[58] Aaron Oord et al. "Parallel wavenet: Fast high-fidelity speech synthesis". In: *International conference on machine learning.* PMLR. 2018, pp. 3918–3926.

[59] Aaron van den Oord et al. "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499* (2016).

[60] George Papamakarios, Theo Pavlakou, and Iain Murray. "Masked autoregressive flow for density estimation". In: *Advances in neural information processing systems* 30 (2017).

[61] George Papamakarios et al. "Normalizing Flows for Probabilistic Modeling and Inference". In: *arXiv e-prints*, arXiv:1912.02762 (Dec. 2019), arXiv:1912.02762. arXiv: `1912.02762 [stat.ML]`.

[62] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32.* Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[63] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[64] Michael Poli et al. "TorchDyn: Implicit Models and Neural Numerical Methods in PyTorch". In: ().

[65] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* 1st. USA: John Wiley and Sons, Inc., 1994. ISBN: 0471619779.

[66] Martin Riedmiller. "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method". In: *European conference on machine learning.* Springer. 2005, pp. 317–328.

[67] Jørgen Nilsen Riseth. "Gradient-Based Optimization in Shape Analysis for Reparametrization of Parametric Curves and Surfaces". 2021.

[68] Murray Rosenblatt. "Remarks on a Multivariate Transformation". In: *The Annals of Mathematical Statistics* 23.3 (1952), pp. 470 –472. DOI: `10.1214/aoms/1177729394`. URL: `https://doi.org/10.1214/aoms/1177729394`.

[69] Dietmar A. Salamon. *Measure and Integration.* EMS Textbooks in Mathematics. European Mathematical Society Publishing House, Zuerich, Switzerland, 2016. ISBN: 9783037191590. URL: `https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1424355&site=ehost-live&scope=site`.

[70] T.B. Sebastian, P.N. Klein, and B.B. Kimia. "On aligning curves". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.1 (2003), pp. 116–125. DOI: `10.1109/TPAMI.2003.1159951`.

[71] Tatiana Shingel. "Interpolation in special orthogonal groups". In: *IMA Journal of Numerical Analysis* 29.3 (July 2008), pp. 731–745. ISSN: 0272-4979. DOI: `10.1093/imanum/drn033`. eprint: `https://academic.oup.com/imajna/article-pdf/29/3/731/2624506/drn033.pdf`. URL: `https://doi.org/10.1093/imanum/drn033`.

[72] David Silver et al. "Deterministic policy gradient algorithms". In: *International conference on machine learning.* PMLR. 2014, pp. 387–395.

[73] Matthijs T. J. Spaan. "Partially Observable Markov Decision Processes". In: *Reinforcement Learning: State-of-the-Art.* Ed. by Marco Wiering and Martijn van Otterlo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 387–414. ISBN: 978-3-642-27645-3. DOI: `10.1007/978-3-642-27645-3_12`. URL: `https://doi.org/10.1007/978-3-642-27645-3_12`.

[74] Anuj Srivastava et al. "Shape Analysis of Elastic Curves in Euclidean Spaces". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.7 (2011), pp. 1415–1428. DOI: `10.1109/TPAMI.2010.184`.

[75] Zhe Su, Eric Klassen, and Martin Bauer. "The Square Root Velocity Framework for Curves in a Homogeneous Space". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).* 2017, pp. 680–689. DOI: `10.1109/CVPRW.2017.97`.

[76] Richard S Sutton. *Reinforcement learning : an introduction.* eng. Second edition. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2018. ISBN: 978-0-262-03924-6.

[77] Esteban G Tabak and Cristina V Turner. "A family of nonparametric density estimation algorithms". In: *Communications on Pure and Applied Mathematics* 66.2 (2013), pp. 145–164.

[78] Esteban G. Tabak and Eric Vanden-Eijnden. "Density estimation by dual ascent of the log-likelihood". In: *Communications in Mathematical Sciences* 8.1 (2010), pp. 217 –233. DOI: `cms/1266935020`. URL: `https://doi.org/`.

[79] Howard M. Taylor and Samuel Karlin. *An Introduction To Stochastic Modeling.* Third edition. Academic Press, 1998.

[80] Dustin Tran et al. "Discrete Flows: Invertible Generative Models of Discrete Data". In: *CoRR* abs/1905.10347 (2019). arXiv: `1905.10347`. URL: `http://arxiv.org/abs/1905.10347`.

[81] J.N. Tsitsiklis and B. Van Roy. "An analysis of temporal-difference learning with function approximation". In: *IEEE Transactions on Automatic Control* 42.5 (1997), pp. 674–690. DOI: `10.1109/9.580874`.

[82] Pavan K Turaga et al. *Why Use Sobolev Metrics on the Space of Curves.* 1st ed. 2016. Cham : Springer International Publishing : pp. 233–255. ISBN: 3-319-22956-7.

[83] Cédric Villani. *Optimal Transport: Old and New.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. ISBN: 978-3-540-71050-9. DOI: `10.1007/978-3-540-71050-9_1`. URL: `https://doi.org/10.1007/978-3-540-71050-9_1`.

[84] Prince Zizhuang Wang and William Yang Wang. "Riemannian Normalizing Flow on Variational Wasserstein Autoencoder for Text Modeling". In: *CoRR* abs/1904.02399 (2019). arXiv: `1904.02399`. URL: `http://arxiv.org/abs/1904.02399`.

[85] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3 (1992), pp. 279–292.

[86] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989).

[87] Lilian Weng. "Flow-based Deep Generative Models". In: *lilianweng.github.io* (2018). URL: `https://lilianweng.github.io/posts/2018-10-13-flow-models/`.

[88] Marco Wiering and Martijn Van Otterlo. *Reinforcement Learning: State of the Art.* English. Springer, 2012. ISBN: 978-3-642-27644-6. DOI: `10.1007/978-3-642-27645-3`.

[89] Esten Nicolai Wøien. "A Semi-Discretized Method for Optimal Reparametrization of Curves". 2019.

Appendix A

Implementation details

A.1 Deep Q-learning

The code used in the experiments related to deep Q-learning can be found at `https://github.com/alexarntzen/neural-reparam`. The optimization procedure and neural network implementation were done with the PyTorch [62] library. The networks were initialized with the PyTorch default distribution, and the optimization method used was Adam [39]. Moreover, to enable further experiments, this code also contains an implementation of reparametrization MDPs using the Gym framework [10].

A.2 Normalizing flows

The code used in the experiments related to normalizing flows can be found at `https://github.com/alexarntzen/shapeflow`. The optimization procedure and neural network implementation were done with the PyTorch [62] library. The networks were initialized with the PyTorch default distribution, and the optimization method used was Adam [39]. The invertible ResNets used in this thesis used code originally written for [16]. Continuos normalizing flows used the Neural ODE solver package TorchDyn[64], which is an extension of code written for [15].