LEARNING LOW-DIMENSIONAL LATENT REPRESENTATIONS OF DEMONSTRATED TRAJECTORIES FOR ROBOTS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Travers Rhodes

August 2024

LEARNING LOW-DIMENSIONAL
LATENT REPRESENTATIONS OF
DEMONSTRATED TRAJECTORIES
FOR ROBOTS

Travers Rhodes, Ph.D.
Cornell University 2024

For robots to perform intricate manipulation skills, like picking up a slippery banana slice with a fork, it is often useful to have a human demonstrate how to perform that skill for the robot. Humans can perform the desired motion multiple times in front of the robot, and the robot can record the demonstrated trajectories and build a model of the demonstrations. If the robot can learn a good model of the different ways to perform the desired motion, the human and the robot can then work together to pick a trajectory for the robot to perform to solve the task. This dissertation investigates the machine learning component of that example: ”How can a robot learn a good model of demonstrated trajectories?” We present multiple advances in the ability of robots to model demonstrated trajectories using latent variable models. These approaches include better model regularization to take advantage of the small size of datasets of human demonstrations, better architectural choices to separate the timing and spatial variations of the demonstrated trajectories, and an investigation into how to disentangle the meaning of the variables in the latent variable model. Theoretical justifications for the contributions are presented alongside empirical evaluations performed on a physical robot arm.

BIOGRAPHICAL SKETCH

Travers Rhodes grew up in Hanover, NH, and graduated from Hanover High School in 2008. He received his bachelor's degree in Physics and Mathematics from Harvard College in 2012 and his master's degree in Robotics from Carnegie Mellon University in 2017, working under the supervision of Manuela Veloso. He joined Cornell to pursue his PhD in Computer Science in 2019.

To Taylor

ACKNOWLEDGEMENTS

First, I would like to thank my incredible advisor, Daniel Lee. One of the best parts of working on my PhD has been the research meetings I've had with Dan where I've come to him with disheartening results or a difficult choice for research direction, and he has consistently helped me make a good choice, and even more importantly has invariably made me feel excited about the direction of our future work and helped me look forward to implementing it. It has also been particularly lucky that I got to have a PhD advisor who happens to be almost always right about everything and also has the patience to give me a week or two or three to explore and think and realize on my own that he was right the whole time. I also want to thank my committee members, Tapomayukh Bhattacharjee, Ziv Goldfeld, and Guy Hoffman, who put up with slow and steady changes in my overall research direction and sudden surprise administrative forms. I have thoroughly enjoyed my research discussions with each of you and am honored to have worked with you during my PhD.

I've been very lucky to participate in the research community created by many generous individuals at Cornell. In NYC, I've gotten to share in the communal division of snacks, meals, and research equipment fostered and inspired by the comraderie of Ilan Mandel, and I've enjoyed the camaraderie of my fellow PhD students over chips and soda at the campus-wide happy hours (where we human-robot interaction and human-computer interaction researchers learned more about "human interaction"). I've gotten to enjoy the hard-working espresso machine with fair-trade coffee beans from around the world provided generously by Wendy Ju, and I've gotten to have wine that even the most élite would be impressed by at lab dinners hosted by Dan Lee. I want to especially thank the other first-year students in my cohort for the fun times in our basement office during my time at the Ithaca campus. Thank you to Hadas Kress-Gazit and her students in the Verifiable Robotics Research Group, the students in the Robotic Personal Assistants Laboratory, and Tapomayukh Bhattacharjee and his students for letting me attend your lab meetings (remotely, as needed) and making me feel like part of a big Robotics family at Cornell. This reminds me to thank Ken Birman for being my "Faculty Mentor" when I first joined Cornell, helping me navigate the transition to the Cornell Tech campus in NYC and any other incidents that came up during my first semester at Cornell. I offer special thanks to Bobby Kleinberg, Becky Stewart, and RVR for their help in navigating all the academic, bureaucratic, and logistical aspects of my PhD. It's been a pleasure to have very helpful and sympathetic administrators. Thank you to Tamika Morales, Joe Tangredi, and the whole facilities team at Cornell Tech for helping us build a great lab space. The author would like to thank Chanwoo Chun, Ge Gao, Niko Grupen, Sasha Rush, Sloan Nietert, David Goedicke, Rei Lee, Ilan Mandel, Frank Bu, Matt Franchi, Teresa Parreira, Natalie Friedman, Alexandra Bremers, Stacey Stone, Kyon Terao, Saki Suzuki, Hauke Sandhaus, Tauhid Tanzim, Amazia Thompson, Valts Blukis, and everyone I forgot to list for many insightful conversations.

On a more personal note, I want to thank my supportive friends and family for helping me see that my work was still worth doing, especially during the COVID-19 pandemic when my outdoor walks would pass by the temporary field hospital and morgue set up in Central Park. Thank you to Adam Krahn, Melissa Theiss, Dylan Matthews, and Adam Kind for regularly scheduled remote hangout sessions to watch TV and play Mario Kart, online bridge, and chess. Thank you to Sabrina Lee and Ian Kumekawa for organizing the Woody Fan Club. Thank you to Adrian Veres and Aubrey Faust for the quarterly checkins on goals and progress and for helping me with my applications to graduate school in the first place. Thank you to Becky Lubin for being my accountability buddy and receiving months of emails saying I was at my desk and had started working. Thank you to Griffin Berlstein for organizing the Sci-fi Book Club. Thank you to my in-laws for making a lovely place to live during my years as a student in NYC.

A dissertation only happens if the right support network and education have gotten you accepted to a PhD program in the first place. Thank you to my dad for giving me a head start on learning math and computer programming. Thank you to my brother for telling me I was especially good at math and science (and for pretending he wasn't). Thank you to my grandmother Janey Symington for being my "brag-buddy" and reveling in all my academic accomplishments, to my grandmother Mae Rhodes for thinking that "of course" I would get a PhD, and to my grandfathers Stuart Symington, Jr., and C. Harker Rhodes, Jr., for being happy for me whatever I did. I am grateful to Dr. Susan Frankenstein at the Army Corps of Engineers and Prof. Nigel Glover at Durham University for my first research jobs. Thank you to Andrew Fedorchek and all my bosses at Applied Predictive Technologies for taking a chance on me and paying me to be a Software Engineer. Thank you to Edward Rhodes for (perhaps jokingly) saying that, since he gets paid to teach, he's a big fan of people being students for as many years as possible. Thank you to Manuela Veloso for welcoming me back to academia, being an incredible master's advisor, and using her influence to get me accepted to Daniel Lee's lab for my PhD.

Finally, I want to thank my wife, C. Taylor Poor. Thank you for saying my work was good and important (even when I didn't believe you) and for the cumulative hours of proofreading.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivational Example

We begin this dissertation with a motivational example of an idealized collaborative human-robot team. Imagine the human agent wants a robot arm to pick up a slippery banana slice using a fork. Imagine further that the robot doesn't know how to accomplish this task. Penetration and contact friction are particularly challenging to model, so it's unlikely the robot has already trained a banana slice acquisition policy using reinforcement learning in simulation. To teach the robot, the human could demonstrate the desired motion to the robot so that the robot can learn how to perform the task, using the approach known as "learning from demonstration." Suppose the human demonstrates how to pick up the banana slice multiple times, for example, by approaching the banana slice from different directions or holding the fork at different angles. In that case, the robot can learn multiple (infinitely many) different ways to pick up the banana slice by building a continuous model of the demonstrated motions. Given that model of ways to pick up a banana slice, the human and robot could work together in the future to generate new trajectories to pick up banana slices in new scenarios. The human could search the robot's model of trajectories and pick which modeled version of the motion the robot should execute. Fig. 1.1 shows this idealized system.

As part of this learning-from-demonstration approach, the robot learns a generative model of trajectories to pick up banana slices. This learned model can generate new trajectories that the robot can execute in new scenarios. In

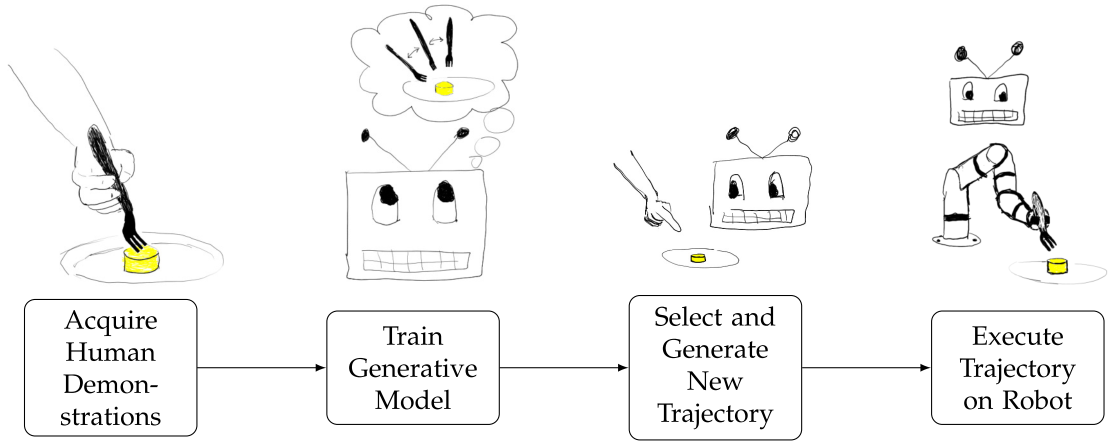

Acquire Human Demon-strations
Train Generative Model
Select and Generate New Trajectory
Execute Trajectory on Robot

Figure 1.1: Idealized component diagram of a learning from demonstration approach. A sample of human demonstrations is collected. The robot learns a generative model of how to perform the skill. The user prompts the robot to generate some trajectory to accomplish the desired task and validates that trajectory. The robot then executes the newly generated motion and accomplishes the task.

this dissertation, we focus on how best to learn this generative model of the demonstration data. We consider the class of "latent variable models," generative models where the user can input real numbers to prompt the model to generate novel trajectories. These inputs to the generative model are "latent values." The model has learned a "latent representation" of the demonstration data. Different latent values given to the generative model will output different novel trajectories. The user could use a joystick to search over that low-dimensional space of latent values to find a trajectory they want the robot to execute. This approach of using a latent variable model as a way for the user to select and generate a trajectory is shown in Fig. 1.2. Our research investigates ways to develop better latent variable models for this system.

With this motivational example of why we want to train good latent variable models, we now briefly introduce latent variable models and explain some general properties that we would prefer our latent variable model to have for our

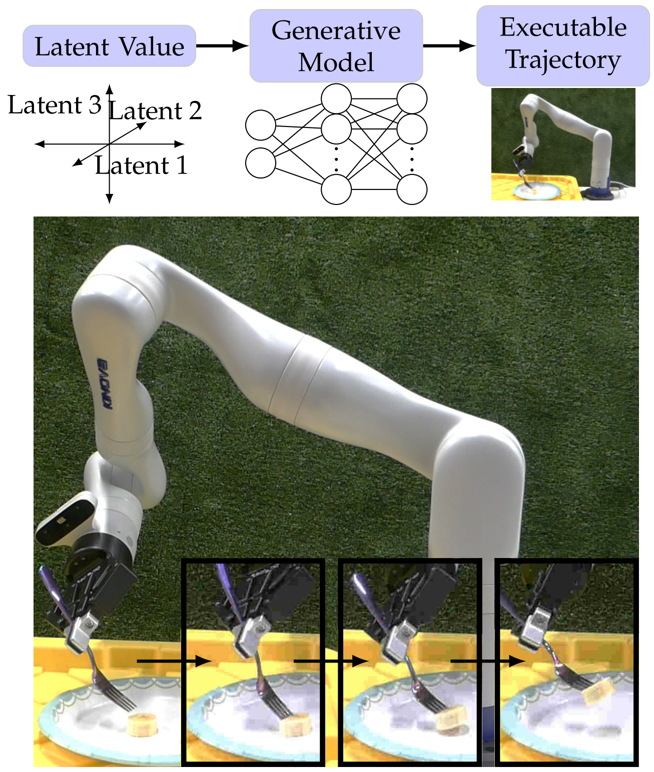


Figure 1.2: Runtime execution of the learned model. The user can use a joystick or other input device to search the latent space and select a latent value. The robot passes that latent value into a generative model to create a full trajectory based on the user's selection. The robot executes that selected trajectory.

use case.

1.2 Introduction to Latent Spaces

In the example above, the user selects a latent value to generate a desired trajectory. That latent value contains information about the trajectory the user wants the robot to execute. This section presents some high-level intuition about the meaning of the latent values.

There are many complementary ways to understand the meaning of latent values. Latent values contain compressed information about a specific trajec-

$$\text{The latent space is} \left\{ \begin{array}{c} \text{a compressed representation[6]} \\ \\ \text{a learned manifold[90]} \\ \\ \text{a set of features[33]} \end{array} \right\} \text{of the data}$$

Figure 1.3: A latent value has many purposes, and eliding minor details can be described using any of the terminology shown in this figure.

tory, label the trajectory's location on the manifold of demonstrated trajectories, and model the trajectory's features. Fig. 1.3 summarizes these multiple meanings of a latent value.

These views of the latent space can be useful when designing and analyzing our trained latent variable models. In this dissertation, we consider:

- How efficiently does our latent variable model compress the demonstration data into the lower-dimensional representation?
- How can we regularize the learned manifold to reduce overfitting to the training data?
- What features does our model learn about the training data, and how can we encourage the model to learn certain types of features?

While our motivational example considers trajectory data, a latent variable model can be trained on a wide variety of data, including, for example, images. A generative latent variable model trained on images could be used to generate novel images, and again, the user could use a joystick or other input modality to select a latent value and guide the model in generating certain types of images.

1.3 Improving Latent Variable Models

There are several properties that are good for our latent variable model to have. For example, we might want to make sure that our model is "interpretable" to the user so that, in our idealized example, it is easier for the user to understand the meaning of the latent values so they can search for a good latent value. Additionally, we may want certain directions in the latent space to correspond to particular features of the data. If each latent variable corresponds to a different factor of variation of the data, this would be a "disentangled" model.

Though it remains an open question how best to define the "interpretability" of a latent space, we can nevertheless provide examples to give intuition about what we mean by "interpretability" in this dissertation. For example, consider the latent space of images of simple pictures of cars and buildings in Fig. 1.4. That figure shows images generated from different latent values sampled at a uniformly spaced grid of points in a two-dimensional latent space. That latent space is interpretable because the z_1 latent variable can be easily understood as corresponding to the position of the car, and the z_2 latent variable is understood to correspond to the position of the building. Because that interpretation is simple, the model is interpretable.

A generally stronger notion of interpretability is disentanglement. In a "disentangled" latent space, the independent factors of variation that generated the latent space are separated into different latent variables. To assess whether a model is disentangled, one needs to know what independent factors of variation generated the data. Models can fail to generate disentangled latent spaces if they instead learn rotated latent spaces, like the one in Fig. 1.5. If the true in-

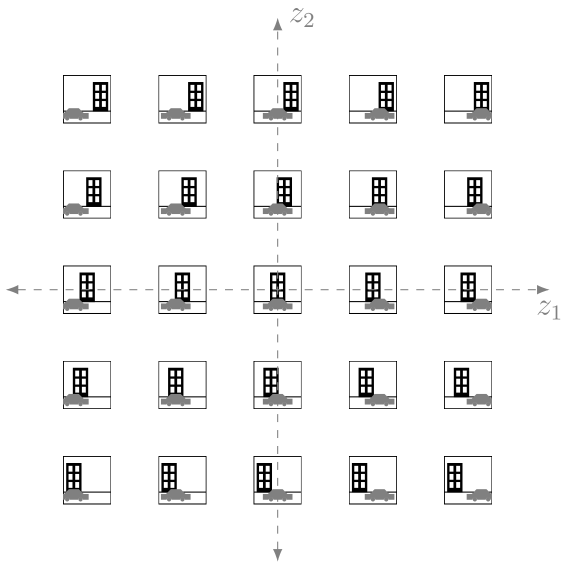


Figure 1.4: Though the best definition of "interpretability" is still an open question in research, we can think of this depiction of a latent space of images as "interpretable" because we can easily understand what the different latent directions correspond to. In particular, the z_1 direction corresponds to the car's position, and the z_2 direction corresponds to the building's position.

dependent factors of variation that generated the data are the car position and the building position, then the new latent space shown in Fig. 1.5 might still be "interpretable" (because we can understand the z_1 direction to correspond to the average position of the car and building, and the z_2 direction to correspond to how far the building is to the right of the car). However, it would not be "disentangled" if the true factors of variation were the car and building positions because the latent values correspond to a combination of the independent factors "car position" and "building position."

1.4 Autoencoder Variants

This dissertation considers a class of latent variable models known as autoencoders. Autoencoders are a latent variable model with an "encoder" that en-

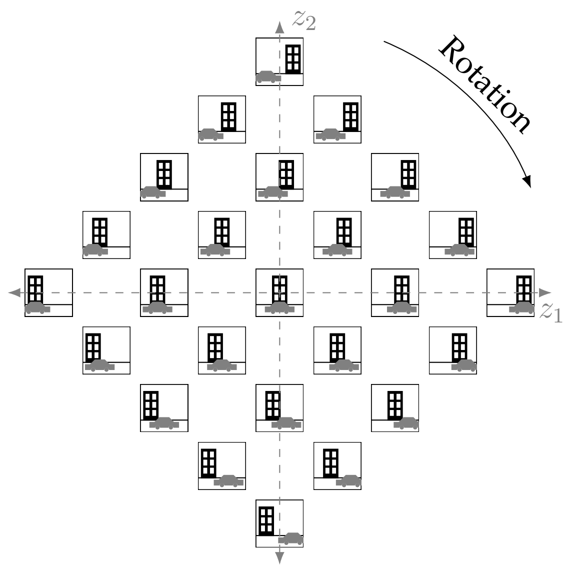


Figure 1.5: If the true independent factors of variation that generated the image dataset were the position of the car and the position of the building, then this latent space would not be considered "disentangled" because the individual latent values correspond to a combination of the independent factors "car position" and "building position."

codes demonstration data into a latent value and a "decoder" that takes in a latent value as its input and outputs a datapoint. The "decoder" is what we used in the example above to map from the latent space (real numbers selected by the user using a joystick) to generated trajectories (which can then be executed by the robot to pick up a banana slice). The decoder is also what we used in the example above to show how a grid of values in the latent space maps to different generated images (images with the car and building in different locations). The encoder goes the other way, taking in a datapoint and returning a value in the latent space. The autoencoder architecture is visualized in Fig. 1.6. In this section, we discuss the mathematical formulation of various types of autoencoders.

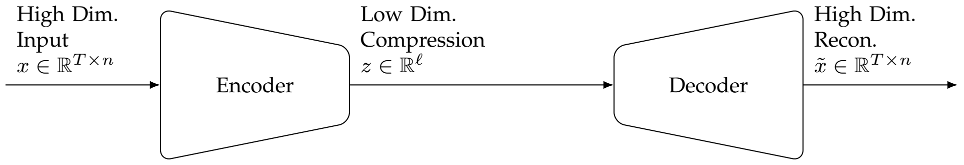

High Dim.
Input
$x \in \mathbb{R}^{T\times n}$
Encoder
Low Dim.
Compression
$z \in \mathbb{R}^{\ell}$
Decoder
High Dim.
Recon.
$\tilde{x} \in \mathbb{R}^{T\times n}$

Figure 1.6: This diagram visualizes the latent representation as a compression of trajectory data. It takes in the high-dimensional trajectory on the left, encodes it as a low-dimensional representation in the middle, and then decodes it back to a high-dimensional reconstructed trajectory on the right.

1.4.1 Autoencoder

The simplest autoencoder learns the latent space by parameterizing the encoder f and decoder g and then estimating the parameters of those functions that minimize the average over datapoints x of:

$$\underbrace{\|x - g(f(x))\|^2}_{\text{Reconstruction Loss}}$$

In the case of linear g and f, learning an autoencoder is equivalent to performing PCA [69][81]. For more complicated functions, we can learn arbitrarily complicated latent spaces.

Fig. 1.7 visualizes what a learned autoencoder might look like using pictures of cars and buildings. In this hypothetical example, the only thing that varies in the image is the positions of the car and the building, so it should be possible to learn a two-dimensional latent representation where z_1 corresponds to the position of the car and z_2 corresponds to the position of the building. The encoder should take in an image of a car and a building and give the positions of the objects in the image. The decoder takes that two-dimensional latent value and returns an image of a car and a building in the locations corresponding to the latent value.

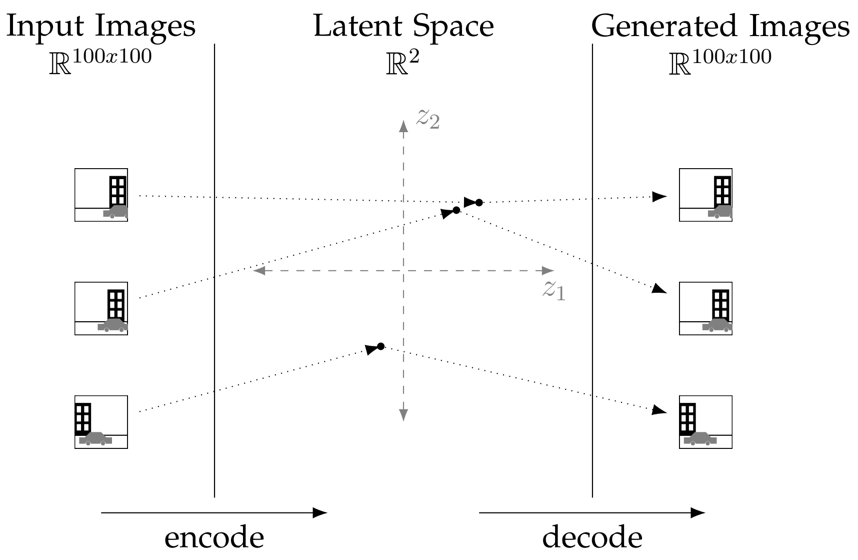


Figure 1.7: An autoencoder of images could look like this, where the input data on the left is high-dimensional image data (with dimensionality on the order of the number of pixels times the number of color channels), and the encoder compresses the image into a low-dimensional latent value (here, represented as a two-dimensional latent value). The decoder can generate a new image from any latent value.

That example describes a very well-behaved autoencoder model with easy-to-interpret latent values and close-to-perfect reconstruction error. However, just having a close-to-perfect reconstruction error for an autoencoder model does not necessarily mean it is easy to interpret. Consider an autoencoder model that takes those same training images of cars and buildings and encodes them into a one-dimensional latent variable $z_1 \in \mathbb{R}$ (instead of our two-dimensional latent space we had before). If we allow the autoencoder encoder and decoder functions to be arbitrarily complicated, and allow z_1 to be an arbitrarily high precision number, then we can achieve arbitrarily good reconstruction error even though we only use a one-dimensional latent space. If the image is an 8-bit grayscale image of 100 by 100 pixels, then a lossless-compression example of such a function would be to have the encoder take the 8-bit grayscale intensities of each pixel in the image, concatenate those values into a single 80,000-bit number and represent that as our single latent variable value $z_1 \in \mathbb{R}$. The decoder could then be the inverse function, reading off every 8 bits from that very long number as the corresponding pixel intensity. While this example may seem a bit contrived, similar things can happen in neural networks, where the learned latent space can be mathematically very complicated and unintuitive but can still give near-perfect reconstruction accuracy on the training data. This problem is addressed in the next section.

1.4.2 Variational Autoencoder (VAE)

A variational autoencoder (VAE) is an unsupervised machine learning approach that models the main factors of variation of the data [48]. The VAE includes an embedding function that takes in a datapoint and returns an associated lower-dimensional latent representation, corresponding to the main factors of variation of the data while removing noise. The VAE also includes a data generation function that takes in a latent value and generates an associated synthetic datapoint. The VAE is trained so that the data generation function creates synthetic data that looks like the training dataset and maintains the main factors of variation in the data. Unlike a simple autoencoder, the VAE uses a cost that tries to ensure nearby points in the latent space generate similar synthetic data points. This cost helps prevent the VAE from learning the very complicated type of encoder described at the end of the previous section.

Specifically, the VAE training loss contains a noisy reconstruction loss, which is computed by embedding a training point into the latent space, perturbing that latent embedding according to a noise distribution, reconstructing back from the latent space into the data space, and then computing the error between the reconstructed data point and original training point. This procedure is visualized in Fig. 1.8, where we show input data being encoded, having noise added to the latent values, and then decoded. The noisy reconstruction loss encourages the VAE to act like an autoencoder and learn an invertible encoding, while simultaneously regularizing the encoder and decoder functions and preventing them from learning arbitrarily complicated functions. In particular, as compared to the example given at the end of the last section, this noise term (related to the size of the "information bottleneck" [3]) gives a theoretical guarantee that the autoencoder now cannot use arbitrarily high-precision latent-space values to encode arbitrarily large amounts of information in the latent values.

Generating a datapoint $\tilde{x} \in \mathbb{R}^n$ from a trained VAE consists of sampling a latent variable $z \in \mathbb{R}^l$ from a standard multivariate Gaussian distribution $N(\mathbb{0}, \mathbb{1})$ and applying a generator function $g : \mathbb{R}^l \to \mathbb{R}^n$ to z to map the latent variable to a generated $g(z)$. If we are training on binary images, then, around this generated image, we assume a Bernoulli probability of similar images $\tilde{x} \sim p(\tilde{x}; g(z))$ with the generated image $g(z)$ as its mean. If we are training on continuous trajectory data, we assume a Gaussian probability distribution $\tilde{x} \sim p(\tilde{x}; g(z))$ with the generated trajectory $g(z)$ as its mean and with predefined constant spherical variance σ_R^2.

Training the VAE uses a multivariate Gaussian embedding distribution $q(z|x)$ for each training datapoint x with mean $h(x)$ (using a learned embedding function $h : \mathbb{R}^n \to \mathbb{R}^l$) and diagonal covariance $\Sigma_{z|x}(x)$ (using a learned

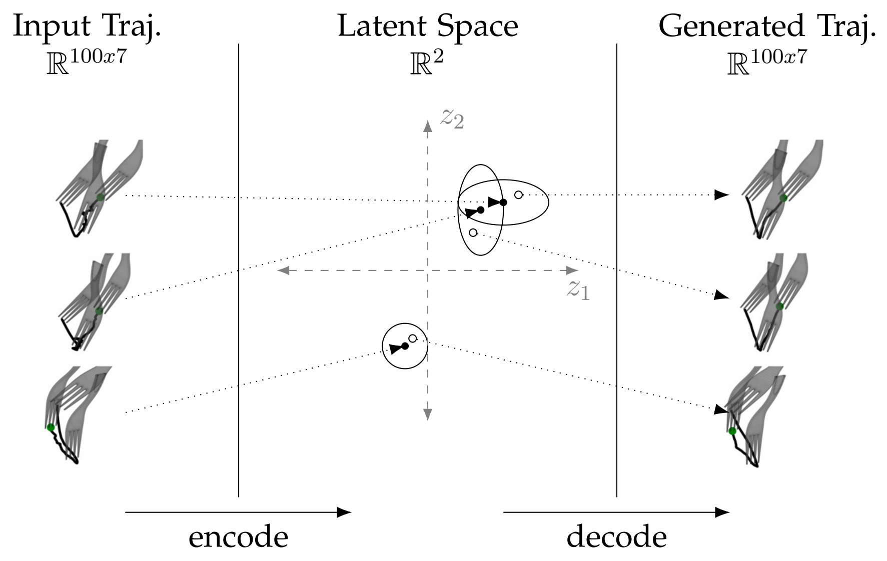


Figure 1.8: A variational autoencoder of trajectories can function as follows, where the input data on the left is high-dimensional trajectory data (with dimensionality on the order of the number of timesteps times the dimensionality of a pose at a single timestep), and the encoder compresses the trajectory into a low-dimensional latent value (here, represented as a two-dimensional latent value). The decoder can generate a new trajectory from any latent value. During training, noise is added to the latent values.

covariance function $\Sigma_{z|x} : \mathbb{R}^n \to \mathbb{R}^l$ that computes the diagonal elements). This embedding distribution $q(z|x)$ is motivated by a desire to approximate the posterior distribution $p(z|x)$.

The objective during training of the VAE is to maximize the Evidence Lower BOund (ELBO), which is defined as $\sum_x L(x)$, where, for each data point x,

$$L(x) = E_{z \sim q(z|x)} \left[p(x; g(z)) \right] - \text{KL} \left(q(z|x) \| N(\mathbb{0}, \mathbb{1}) \right) \tag{1.1}$$

The ELBO is a lower bound on $\log(p(x)) = \log(\int_z p(x|z)p(z)dz)$, which is the likelihood of the data point given our model. Thus, maximizing the ELBO is a proxy for maximum likelihood estimation. The beta-VAE [35] extends the VAE by multiplying the second term in the ELBO by an adjustable hyperparameter β. If we set β to 1, we have the definition of a VAE. The first term in the ELBO is a stochastic reconstruction loss. We sometimes refer to $\Sigma_{z|x}$ as the "embedding noise" since it adds noise to the embedding when estimating the stochastic reconstruction loss.

A standard formulation of beta-VAE is to parameterize the encoder distributions as axis-aligned Gaussians $q(z|x) = \mathcal{N}(e(x), \sigma^2(x))$. Thus, the encoder returns the expected latent value $z = e(x) \in \mathbb{R}^\ell$ for a trajectory, along with the log of the (diagonal) covariance of the encoder noise distribution given by $\log(\sigma^2) \in \mathbb{R}^\ell$. The beta-VAE architecture is shown in Fig. 1.9. Its loss function is $\mathcal{L} = \mathcal{L}_R + \mathcal{L}_{\mathrm{KL}}$, where

$$\mathcal{L}_R = \frac{1}{\sigma_R^2}\mathbb{E}_{x_i,t,\epsilon}\left[\|x_i(t) - f\left(e(x_i) + \epsilon\right)_t\|^2\right] \tag{1.2}$$

$$\mathcal{L}_{\mathrm{KL}} = \beta\mathbb{E}_{x_i}\left[\frac{\|e(x_i)\|^2 + \sum_d(\sigma_d^2(x_i) + \log(\sigma_d^2(x_i))}{2}\right] \tag{1.3}$$

$\mathcal{L}_R$ is a reconstruction loss encouraging the model to encode important information about the trajectories, and $\mathcal{L}_{\mathrm{KL}}$ is a rate regularization, constraining the total amount of information that the model can store about the trajectory [2]. For clarity, we use the subscript x_i to emphasize that these losses are computed as empirical expectations over each training trajectory. The subscript t used in $f(\ldots)_t$ indicates taking the value of the generated trajectory at timestep t (so that it can be properly compared to $x_i(t)$, which is the desired trajectory at that same timestep). σ_d^2 are the elements of σ^2 and ϵ is drawn from a normal distribution with mean 0 and diagonal covariance given by σ^2. β is a regularization hyperparameter.

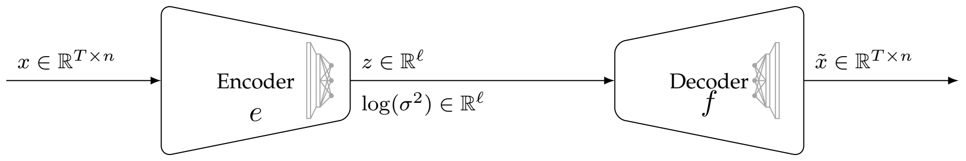

$x \in \mathbb{R}^{T \times n}$
Encoder
e
$z \in \mathbb{R}^{\ell}$
$\log(\sigma^2) \in \mathbb{R}^{\ell}$
Decoder
f
$\tilde{x} \in \mathbb{R}^{T \times n}$

Figure 1.9: The architecture for beta-VAE. Beta-VAE takes in a trajectory x, encodes it into a latent distribution parameterized by z and $\log(\sigma^2)$, and decodes it to a trajectory $\tilde{x}$.

1.5 Alternate Methods

Since our motivational example only uses the generative function (and doesn't need the encoder function), we could also consider generative models that do not directly learn an encoder function. One such approach is the Generative Adversarial Network (GAN) [29]. The GAN approach learns two networks: a decoder network that maps latent values to generated data points (like the decoder in a VAE) and a special discriminator network that is trained to determine whether a datapoint is likely sampled from the training data (a real datapoint) or was generated by the network (a synthetic datapoint). During training, latent values are randomly sampled from the latent space, and corresponding synthetic datapoints are generated using the generative function. The discriminator is then given several real and synthetic datapoints and is trained to try to tell if each point is likely synthetic or real. The generator is trained to try to fool the discriminator by making its generated data look as similar as possible to the real dataset. For simplicity, we do not specifically analyze GANs in this dissertation, but the question of how to do a good job disentangling the latent space also applies to GANs (see, for example, work on InfoGAN [16]), and the model improvements we present in this work could also be applied to GANs. When considering InfoGAN in particular, we note that one of its key contributions was its choice to add a cost function to maximise the mutual information between the observation and a subset of latent variables. The beta-VAE, which forms a basis for our work, also contains a term to maximize the mutual information between the latent variables and the observation, as explained in [9]. While there may be some benefit to only maximizing the mutual information between the observations and a subset of latent variables (as in InfoGAN), instead of between the observations and all latent variables (as in beta-VAE), when comparisons have been made between InfoGAN and beta-VAE-style approaches, InfoGAN does not tend to outperform beta-VAEs [15, 35]. It is important, however, to consider advancements in disentanglement both in the GAN literature as well as in the VAE literature, and to note that our contributions could be used for either architecture.

1.6 Key Contributions

This dissertation considers how to improve latent representations for robotics applications. Our motivational example describes the usefulness of latent variable models for human-robot collaboration. In our example system, a human provides examples to a robot learner, and the robot learns a representation (a latent variable model) of the example data. The human can then select a latent value, and the robot's generative model can create new outputs based on that latent value. If the training data are trajectories, the robot could execute the new, generated trajectory; if they are images, it could display or draw them. The key contributions of this dissertation are:

- Curvature regularization of the latent space. Our curvature-regularized variational autoencoder (CurvVAE) describes a method to regularize the learned manifold better, which helps avoid overfitting the training data. This contribution is described in Chapter 2.

- Separating the timing and spatial latent variables of the latent space. In our TimewarpVAE, we describe a method to separate the timing and spatial features of the latent space of trajectories. This contribution is described in Chapter 3.
- Local disentanglement of objects in latent spaces of images. In our Jacobian L_1 Regularized VAE (JL1VAE), we describe a method to disentangle better factors of variation corresponding to different objects in the latent space of images. This contribution is described in Chapter 4.
- Physical robot experiments. In addition to algorithmic contributions concerning learning from demonstration, we show results of physical robot experiments using our techniques on a Kinova Gen3 robot arm. These physical robot experiments are described in Chapters 2 and 3.

CHAPTER 2

CURVATURE-REGULARIZED VARIATIONAL AUTO-ENCODER

2.1 Learning Latent Representations of Small Demonstration Datasets

In learning from demonstration, a human demonstrates some skill one or more times, and the robot learns from those human demonstrations. Learning from demonstration can speed up robot learning because the robot can take advantage of human expertise. However, one of the challenges of learning from demonstration is the difficulty in collecting expert data. Human demonstrations take time to collect, and therefore, learning from demonstration usually only has a small number of examples, compared to the tens of thousands of examples present in standard machine-learning datasets. For intricate manipulation skills, the learning from demonstration algorithm must be able to learn a complicated model of the skill with many parameters. To be sample efficient, the algorithm must include some form of inductive bias or model regularization.

There are many forms of model regularization. In this chapter, we study a regularization scheme based on curvature. Functions that overfit to training data will tend to have higher curvature, so we penalize the curvature of the learned model in our regularization term. We present a novel learning algorithm, the curvature-regularized variational autoencoder (CurvVAE), which combines the modeling capabilities of a variational autoencoder (VAE) [48] with a curvature regularization term to prevent overfitting to small datasets.

To showcase learning from demonstration using a CurvVAE, we choose an example that requires intricate manipulations and is difficult to simulate. The domain we choose is the acquisition of soft and slippery food items using a fork. In particular, our robot learns to pick up banana slices with a fork, as shown in Fig. 1.2. This problem requires intricate manipulations since simple stabbing motions often cause the food to slip off the fork prongs. It is difficult to simulate this problem because we lack good models of banana slices' deformable nature and the relevant frictional forces. Since humans can demonstrate how to acquire banana slices with a fork, this problem is a good candidate for learning from demonstration, as described in the motivational example in Chapter 1. However, the training dataset is relatively small since learning from demonstration on this problem requires real human examples. Our CurvVAE algorithm can learn a good model of human trajectories in an interpretable way without overfitting.

The contributions described in this chapter[1] are:

- our novel CurvVAE algorithm to learn intricate manipulation skills from demonstrations, including the use of a curvature regularization term to prevent overfitting to small datasets, and
- the application of this algorithm to the manipulation of soft, spearable, slippery food (i.e. banana slices), including evaluation on a physical robotic system resulting in performance better than current state-of-the-art [25].

[1]The work in this chapter was published as [75]

2.2 Related Work on Learning Representations of Small Trajectory Datasets

2.2.1 Trajectory Representation

In the robotics context, trajectories are often represented by key points of splines [63, 4, 73] or by parameters of dynamic movement primitives (DMPs) [73, 62, 72, 22]. While it is easy to time-warp and change the goal position of DMPs, the full set of parameters in both DMPs and splines is generally too large to be conveniently modified by hand for trajectories of robots with multiple joints. By contrast, our method learns an intuitive, very low-dimensional representation of trajectories with human interpretability in mind.

Our architecture is flexible. We could have used the parameters of splines or DMPs as the input to our CurvVAE. Such an approach would be similar to [62], which uses the DMP parameters as the input to principal component analysis (PCA) to learn the natural variation of the data. CurvVAE discourages curvature but does not force exact linearity like PCA does, so our model is more flexible and makes fewer linearity assumptions than the approach presented in [62].

Some authors have trained task-specific representations, which are functions from task variables, like the height of a known obstacle or the location of a dartboard target, to DMP parameters [62, 22]. However, this type of task-parameterized DMP is a supervised problem where the desired representation, namely the task variable, is known *a priori*. Our method is based on the unsupervised learning of the representation of trajectories.

2.2.2 Curvature Regularization

Considering models with regularization terms related to curvature, we note important differences between our work and previous work. In [64], a method is proposed to regularize the Hessian (multivariate second derivative) and equate that regularization with the regularization of a function's "curvature profile." However, the extrinsic curvature of a manifold and the Hessian are not equivalent. Scaling the input space can significantly reduce the Hessian without affecting the extrinsic curvature of the output manifold. We want our curvature regularization term to penalize the curvature of the learned manifold, not to encourage a particular scaling of the latent space. The author is not aware of previous work that penalizes the extrinsic curvature of the learned manifold of a VAE.

2.3 Curvature-Regularized VAE

2.3.1 Curvature-Regularized VAE Formulation

VAEs and other neural-network-based machine learning algorithms generally require large datasets. In cases of limited data, models can benefit from regularization. Model regularization helps prevent overfitting the data at the cost of adding inductive bias. However, inductive bias can have benefits, including making the generative model more human-understandable and disentangling latent factors of variation [55]. In this chapter, we explore the use of curvature regularization to improve the sample efficiency of VAEs.

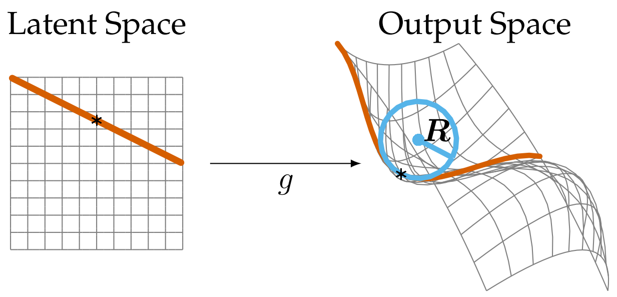


Figure 2.1: After mapping a random point and line through g, we compute the radius R of the best-fit tangent circle to the resulting curve. Our regularization cost is $\frac{1}{R^2}$.

To regularize our model, we penalize the extrinsic curvature of the manifold learned by the generative function. Explicitly, we penalize the square of the reciprocal of the radius of curvature of curves in the learned manifold. The curves are selected by taking random lines through a point in the latent space and mapping them through the function g onto the curved output manifold embedded in the data space, as shown in Fig. 2.1. We average the associated extrinsic curvatures for different randomly sampled points and lines to estimate the overall manifold's curvature. Regularizing the curvature reduces the complexity of the learned model and encourages simpler functions. Additionally, this regularization term encourages interpretable latent variables in the model since it explicitly causes the model to seek out a generator function for which linear changes to the latent space lead to closer-to-linear changes in the output.

For a generator function as g, we use finite differences to estimate the curvature penalty as:

$$\left(\frac{\frac{g(z+h)-g(z)}{\|g(z+h)-g(z)\|+\varepsilon}-\frac{g(z)-g(z-h)}{\|g(z)-g(z-h)\|+\varepsilon}}{\|g(z+h)-g(z)\|+\varepsilon}\right)^2$$

where z is randomly sampled from the latent space, h is randomly sampled from a spherical shell with small radius, and ε is a small constant (ε =1e-10) to avoid divide-by-zero errors. Since h is randomly sampled from a spherical shell, replacing h with $-h$ in the definition above leads to an equivalent curvature estimate. Though the experiments presented here did not include this modification, there may be an additional benefit to instead using the average of $\|g(z+h)-g(z)\|$ and $\|g(z)-g(z-h)\|$ in the denominator, to reduce the noise of this curvature estimate further. This cost is scaled by a hyperparameter γ. We call the beta-VAE architecture with this curvature regularization a curvature-regularized variational autoencoder (CurvVAE). The full CurvVAE training loss is thus the sum over each training data point x of:

$$\|x-g(z)\|^2+\beta\frac{\|f(x)\|^2+\sum_d(\sigma_d^2(x)+\log(\sigma_d^2(x))}{2}$$
$$+\gamma\left(\frac{\frac{g(z+h)-g(z)}{\|g(z+h)-g(z)\|+\varepsilon}-\frac{g(z)-g(z-h)}{\|g(z)-g(z-h)\|+\varepsilon}}{\|g(z+h)-g(z)\|+\varepsilon}\right)^2$$

where $z=f(x)+\epsilon$, h is randomly sampled from a spherical shell with small radius, g is the generator function, f is the embedding function, $\sigma_d^2(x)$ is the variance of our embedding noise in the d'th dimension, ϵ is sampled from an axis-aligned, multivariate normal distribution with variance in each dimension defined by $\sigma_d^2(x)$, and ε is a small constant to avoid divide-by-zero errors.

2.3.2 CurvVAE Example on Fork Poses

To visualize the effect of the curvature regularization term in the CurvVAE loss, consider a sample dataset of fork poses. These poses are the combined 9,920 poses in the fork trajectories described in Section 2.4.1. Each fork pose has three spatial dimensions and four quaternion dimensions and is expressed as a vector of dimension 7. We expect that the sampled fork poses can be well-approximated by a lower-dimensional manifold reflecting the largest factors of variation in the pose dataset.

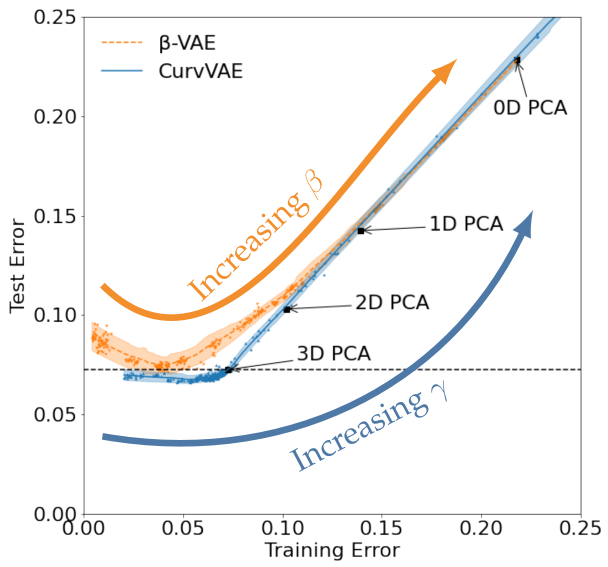


Figure 2.2: Beta-VAE is not able to attain as good a test error as our CurvVAE model. Our CurvVAE also outperforms PCA.

Two hundred randomly sampled poses are used as a very small training dataset, and the rest are the test set. Thirteen different β values are used for beta-VAE and 16 different γ values are used for CurvVAE (holding β at 1e-5). All models use three latent dimensions. For each hyperparameter set, we train ten different models at three different learning rates for a total of 390 different beta-VAE models and 480 different CurvVAE models.

After training each model, the root mean squared error on the training and test sets is plotted. These results are shown in Fig. 2.2, along with the results for PCA models on zero through three dimensions. No value of β for the beta-VAE can achieve as low a test error as the optimal CurvVAE. Although CurvVAE regularizes to encourage a more linear model, it does not require a linear model, so CurvVAE can achieve a lower test error than PCA.

2.4 CurvVAE Machine Learning Methods

Since CurvVAE allows us to train a VAE on a relatively small dataset, we can use a CurvVAE for robotic learning from demonstration in cases where human demonstration data is limited. As an example, we learn a generative model of human fork trajectories in acquiring food. We then show that the learned motions have interpretable latent spaces and lead to effective trajectories when run on a physical robot arm.

2.4.1 Human-Demonstrated Trajectories and Data Processing

We use recordings of human participants picking up banana slices with a fork as our training demonstrations [8]. That fork trajectory dataset contains detailed fork poses (both positions and orientations) for real food acquisition and delivery trajectories in a pretend assistive feeding environment. In the recordings, participants move a fork to a plate, pick up a banana slice with the fork, and then pretend to feed it to a mannequin.

For this work, we just want to model the part of the trajectory where the user is picking up the food, so we extract the individual banana slice acquisition components from the longer demonstration trajectories. However, to learn a continuous model of the trajectory styles, we remove trajectories that are categorically different from others, including those with outlier fork orientations. For the same reason, trajectory sequences from what appears to be a left-handed participant and those in which the force-torque sensor on the fork doesn't detect food impact were removed. Trajectories were truncated to start at most one

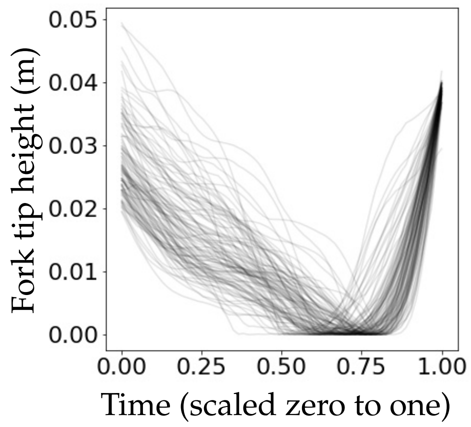


Figure 2.3: Example fork tip height over time in cleaned training data

centimeter above food impact and to end at most four centimeters above the plate to focus the data on the manipulation strategy used to pick up the food and bring it to a stable fork position. After data pre-processing, the dataset has 155 banana acquisition examples.

Each trajectory sequence is linearly retimed to start at time zero and last one arbitrary time unit. Additionally, each trajectory is translated so that the maximal force recording from the trajectory occurs at the X/Y origin, and the lowest height recorded during the trajectory occurs at Z=0. These two steps align trajectories, and trajectories can be again translated or retimed during replay. Fig. 2.3 shows the resulting fork tip height over scaled time after this alignment process. We mean-center the poses and then multiply all the positions by a scaling factor of 5.6 (computed to make the largest variance in the X, Y, or Z direction across the whole dataset equal to one) and multiply all the quaternion orientations by 0.90 (computed so that rotations around the fork tip which move the handle one centimeter are penalized roughly the same amount as translations which move the fork tip one centimeter). When the model is applied and poses are generated for the robot to execute, these centering and scaling transformations are inverted.

2.4.2 Neural Network Architecture for Trajectories

Our trajectory generator function g takes as input a three-dimensional latent value and an additional time parameter. It outputs a pose for that trajectory at that associated time. The pose output by the model is represented as a vector of length seven (three position values and four quaternion values). The generator is a neural network with a single hidden layer of a thousand nodes and uses rectified linear units as its nonlinearities. A shallow but wide architecture is a simple architecture for learning continuous functions. This architecture can generate a full pose trajectory for the fork by sweeping the time parameter between zero and one.

The embedding function f has a shared hidden layer with a thousand nodes and two linear heads attached, one to predict the mean embedding z, and one to predict the log variance of the embedding $\ln(\sigma^2)$, again using rectified linear units for nonlinearities. We follow the standard assumption of axis-aligned Gaussian embeddings so that z and $\ln(\sigma^2)$ are both three-dimensional vectors for a three-dimensional embedding latent space.

During training, paired poses and their associated timestamps (x_a, t_a) and (x_b, t_b) are randomly pulled from a single training trajectory. x_a and x_b are seven-dimensional (position and quaternion) vectors; t_a and t_b are their respective scalar timestamps. We embed (x_a, t_a) using our embedding function f to get a three-dimensional latent embedding z_a and a log-variance estimate $\ln(\sigma_a^2)$. We add Gaussian noise to the mean estimate following the distribution defined by $\sigma_a^2 = \exp(\ln(\sigma_a^2))$ to get the noisy embedding $\tilde{z}_a$. This noisy embedding should be an approximate latent representation of the entire trajectory associated with (x_a, t_a), from which (x_b, t_b) was also drawn. We now pair this embedding with timestamp t_b and use the generator function g to compute a reconstructed pose for $(\tilde{z}_a, t_b)$ with the intent that it be a good estimate of x_b. The reconstructed pose is denoted by $\tilde{x}_b = g(\tilde{z}_a, t_b)$. The squared error between $\tilde{x}_b$ and x_b is our reconstruction loss. This training architecture is shown in Fig. 2.4.

Even though the generator function g is now additionally parameterized by time, the curvature loss calculations only include perturbations of the latent value, keeping the randomly sampled time parameter constant within each curvature loss calculation. The curvature of the individual trajectories over time is not penalized; the penalty is only applied to how trajectories change as the latent value is varied.

2.4.3 Training Hyperparameters

We trained a CurvVAE model using the above model architecture on training data of forks picking up banana slices. We chose hyperparameters based on the tradeoff between latent space interpretability and reconstruction accuracy. Lower β ensures good reconstruction accuracy, but too small a β winds up embedding the latent points very far from each other in the latent space. For our training dataset, $\beta = 0.001$ was a good compromise between these goals. Increasing γ leads to latent traversals that are closer to linear in the output space at the cost of lower reconstruction accuracy. For our training dataset, $\gamma = 0.001$ led

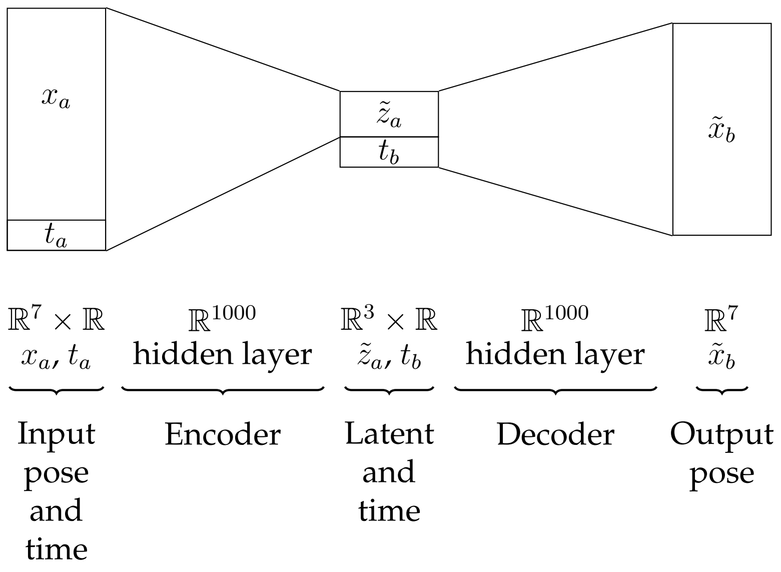


Figure 2.4: During training, a pose $x_a \in \mathbb{R}^7$ from a trajectory is concatenated with its associated timestep t_a and encoded into a distribution parameterized by z_a and $\ln(\sigma_a)^2$ by means of a neural network with a single hidden layer and with two heads (one for z_a and one for $\ln(\sigma_a^2)$). A sample is taken from that encoded distribution to give a noisy encoding $\tilde{z}_a \in \mathbb{R}^3$, which represents a latent representation of the full trajectory. That latent is concatenated with a new timestep t_b and passed through the decoder to give a reconstruction estimate $\tilde{x}_b$ of the actual pose x_b of the trajectory at timestep t_b.

to interpretable models that still had good reconstruction accuracy. We trained for 3,000 batches of 256 pairs of poses randomly sampled from trajectories.

2.4.4 Model Interpretability

The CurvVAE learns an intuitive latent space of trajectories. Changing the latent value leads to interpretable changes to the starting pose of the fork, corresponding to rotations of the fork around different axes. Fig. 2.5 shows the change in fork starting pose as we vary the three-dimensional latent value along different dimensions. The first latent value corresponds to the rotation of the fork's starting pose around the vertical axis. The second latent value corresponds to

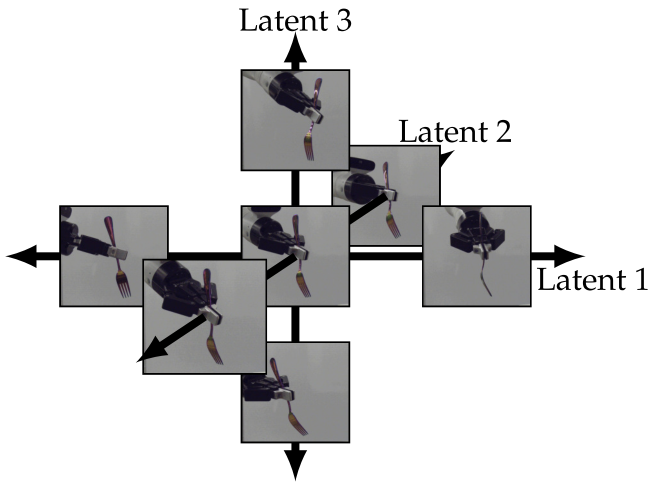


Figure 2.5: Fork starting poses for the CurvVAE trajectory. Each latent direction rotates the fork along a different axis.

tipping the handle away from or toward the viewer. The third latent value corresponds to tipping the handle to the left/right.

2.5 CurvVAE Physical Robot Experiments

2.5.1 The Robot Arm

Experiments are performed on a 7-DOF Kinova Gen3 Arm [49] with a two-fingered Robotiq gripper (85mm stroke width) [80]. A metal fork is held at a calibrated location and orientation in the gripper. We run a 500Hz control loop on an Intel Core i7 CPU running a real-time version of Ubuntu to send effort commands to the arm. The control computer converts high-level desired joint-position commands to low-level effort commands to send to the arm using PID control.

2.5.2 Trajectory-Following Implementation

The target trajectory is encoded as 64 waypoint poses to be executed in 6.4 seconds. We use a Jacobian-based controller to estimate the joint changes required to move the fork-tip held in the robot end-effector from a starting pose through the desired waypoints. These joint changes are scaled as necessary to ensure the robot doesn't move unreasonably quickly, and together, they form a set of desired joint angle waypoints. These desired joint angles are sent to the control computer alongside a desired time-to-goal for each value of desired joint angles. The real-time control computer computes intermediate waypoint joint positions to move each joint linearly from its current actual angle to arrive at the next desired angle at the desired time. PID control is then used to send computed effort commands to the robot arm at 500Hz to follow the intermediate waypoint joint positions.

2.5.3 CurvVAE Latent Value Selection

The latent values of the trained CurvVAE are interpretable, as seen in Fig. 2.5, and correspond primarily to different starting orientations of the fork. We choose latent values that approach the banana from approximately the same cardinal direction and which ensure that no fork prong is tipped closer to the table than the other prongs. Under those constraints, latent values are chosen that tilt the fork away from the banana slice at various angles. In particular, we choose latent values that tilt the fork so that the prongs start 30°, 45°, and 60° off of vertical. These trajectories are visualized in Fig. 2.6.

2.5.4 Baseline Trajectories

We compare the food acquisition efficiency of these learned CurvVAE trajectories to baseline trajectories inspired by [25]. The state-of-the-art trajectory presented in that work holds the fork handle at a $45°$ angle and moves the fork along a $45°$ line to skewer the food. The fork prongs are at a $75°$ angle during motion, so we label this trajectory the $75°$ baseline. We additionally test holding the fork prongs at a $45°$ angle while moving them at a $45°$ angle to skewer the banana slice. After the banana is skewered for both baseline trajectories, the fork attempts to lift it vertically (without changing the orientation of the prongs). We choose these as our baselines because they are easy to code, intuitive, and include the current state of the art of [25].

A beta-VAE model was also trained on the same dataset ($\beta = 0.001$) and a PCA model, both with the same latent dimension of three. Again, latent values were chosen for these models to give similar starting angles as those chosen for the CurvVAE model.

2.5.5 Data Collection

Food acquisition success on banana slices is tested in a controlled laboratory setting. Bananas are sliced to be $\sim$1.5cm thick and placed on a marked location on a plate in a known position relative to the robot base. For each banana slice, we test multiple strategies. By testing multiple different strategies on each slice in a randomized order, we control for variations in banana slices. Each strategy is tested 25 different times, and we count the number of times the food was successfully acquired, where success requires staying on the fork for at least

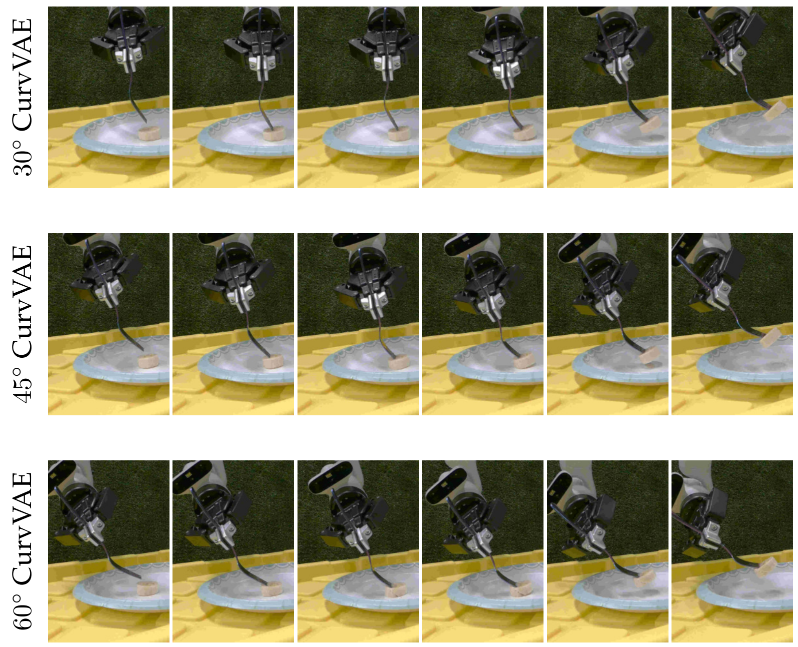


Figure 2.6: Still images from CurvVAE trajectories. Trajectories were generated from the same model using different latent values.

five seconds. Since the type of plate can affect food acquisition performance, the entire experiment was run twice, once using a paper plate and once using a ceramic plate. The ceramic plate had less surface friction, making for a more challenging environment.

2.6 Results and Discussion on Learning Representations of Small Trajectory Datasets

The results of the food efficiency acquisition experiments are presented in Fig. 2.7. Like [25], we see that soft food acquisition can be challenging for the hardcoded trajectories because banana slices are slippery and can slide off the prongs of the fork during lifting. However, the trajectories learned from demonstration are consistently better at picking up banana slices. The difference is

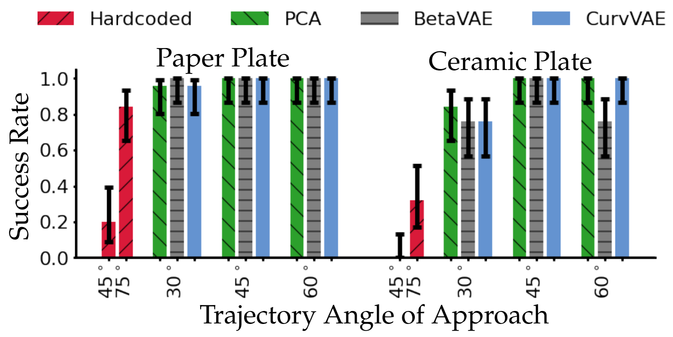


Figure 2.7: Banana slice acquisition success rates.

not statistically significant when the experiment is conducted on a paper plate. However, when the experiment is conducted on a ceramic plate, which has less friction and allows the banana to slide more easily, the performance increase is much larger and is statistically significant ($p < 0.05$).

While this manipulation task suggests that CurvVAE may outperform beta-VAE, in general, this manipulation task is not sensitive enough to differentiate CurvVAE performance from other learning from demonstration strategies.

This chapter showed how robots can learn an intuitive latent space from small datasets using a CurvVAE. We applied the CurvVAE to learning from demonstration and showed that the learned latent space is low-dimensional enough to be meaningfully understood and could, therefore, be directly controlled. We further experimentally validated that the CurvVAE model's generated trajectories have a better success rate than hardcoded baseline trajectories, including the state-of-the-art $75°$ baseline presented in [25].

CHAPTER 3

TIMEWARP VARIATIONAL AUTO-ENCODER

3.1 Temporal and Spatial Variations of Trajectories

Continuous trajectories are inherently infinite-dimensional objects that can vary in complex ways in both time and space. However, in many practical situations, their intrinsic sources of variability can be well-approximated by mapping onto a low-dimensional manifold with points labeled by their latent values. When a human demonstrates trajectories for a robot, it is useful for the robot to learn to model the most expressive latent factors controlling the functionally relevant parts of the demonstration trajectories. For many types of demonstrations, such as in gesture control or quasistatic manipulation, it is highly advantageous to explicitly separate the exact timing of the trajectory from the spatial latent factors. This chapter introduces a method that learns such a representation to generate novel fast trajectories for a robot arm, as shown in Fig 3.1.

Consider the problem of trying to average two samples from a handwriting dataset generated by humans drawing the letter "A" in the air [12]. If we scale two trajectories linearly in time so that their timestamps go from 0 to 1, and then average the two trajectories at each timestep, the resulting average does not maintain the style of the "A"s. This is because the average is taken between parts of the two trajectories that do not naturally correspond. An example of this averaging, with lines showing examples of averaged points, is shown in Fig. 3.2a. Scaling trajectories to have constant speed also doesn't solve the issue (Fig. 3.2b). A common approach like dynamic time warping (DTW) [82] can lead

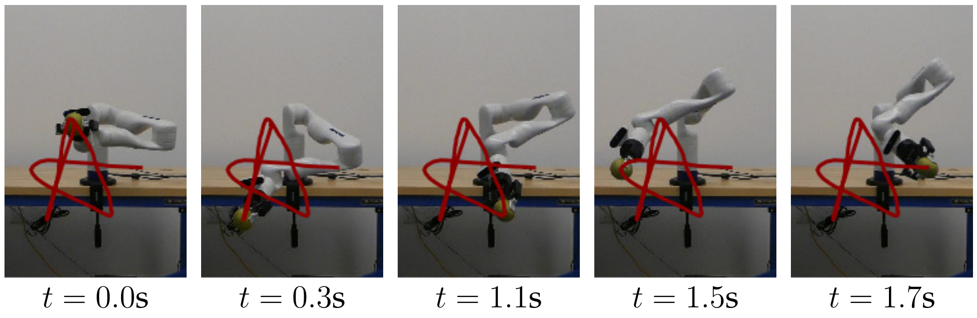
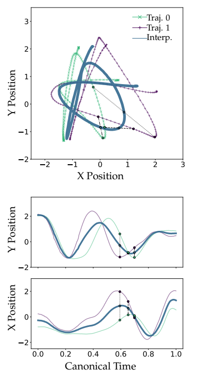


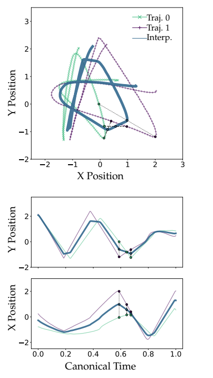
Figure 3.1: TimewarpVAE learns a low-dimensional latent representation of complex trajectories that explicitly factorizes timing and spatial styles. The Kinova Gen3 robot arm can draw various versions of the letter "A" more quickly by speeding up or slowing down different parts of the trajectory to obey dynamical mechanical constraints. The resulting end-effector path is overlaid on the images.

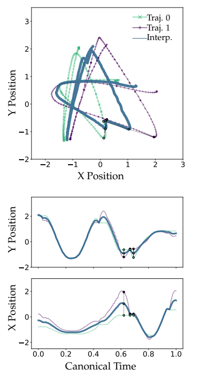
to unintuitive results. When averaging these same two trajectories, DTW only takes in information about these two trajectories and does not use contextual information about other examples of the letter "A" to better understand how to align the timings of these trajectories.[1] In Fig. 3.2c, we use the `dtw` package [27] to align the trajectories before averaging them at corresponding timesteps. We see the resulting trajectory is spatially close to the input trajectories, but it again does not maintain the style of the "A"s.

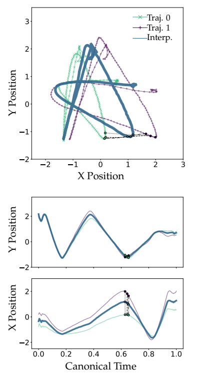
We want to build a representation of trajectories that has a separation of timing and spatial latent variables, so that spatial latent interpolations generate trajectories that nicely spatially interpolate trajectories. Our TimewarpVAE takes the time alignment benefits of DTW and uses them in a manifold learning algorithm to align the timings of similar trajectories. Our results are shown in Fig. 3.2e. The Rate Invariant Autoencoder of [51] is similar in also learning a latent space that separates timing and spatial factors of variation, with the spatial

[1]We use the terms "time-warping," "time alignment," and "registration" interchangeably.


(a) Uniform time scaling


(b) Constant speed scaling


(c) DTW time alignment


(d) Rate Invariant AE

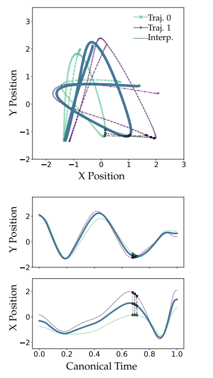

(e) TimewarpVAE (ours)

Figure 3.2: Interpolations in latent space between canonical trajectories using various models. For Rate Invariant Autoencoder and TimewarpVAE, we use a sixteen dimensional spatial latent space and the interpolation is constructed by decoding the average of the spatial latent embeddings. The resulting average trajectory is plotted alongside the reconstructions of the original two trajectories. The Rate Invariant Autoencoder can learn to ignore parts of the canonical trajectory during training, leading to the jittering seen at the beginning and end of the canonical trajectory.

interpolations shown in Fig. 3.2d. Our TimewarpVAE approach has three main contributions[2] that make it better suited for generating robot trajectories than a Rate Invariant AE. The first is the parameterization of the generated trajectory as an arbitrarily complicated neural network function of time rather than basing the trajectory calculation on piecewise linear functions with a pre-specified number of knots. The second is that our approach includes a regularization term, so the model is correctly penalized for extreme time warps. Finally, because of our time-warping regularization term, a robot can optimize the trajectory timing to account for its own joint torque and speed limitations, speeding up its execution of learned trajectories while regularizing to stay close to training timings. An example optimized trajectory is shown in Fig 3.1.

[2]The work done in this chapter was published in [77]

3.2 Related Work on Time-Warping During Representation Learning

Learning a latent space of trajectories that combines the timing and spatial parameters together into a single latent space appears in [88], [17], and [59], among others. TimewarpVAE, like the Rate Invariant Autoencoder of [51], instead separates the timing and spatial latent variables, giving a more efficient spatial model. Our work is an improvement over the Rate Invariant AE in several important ways. First, our work is not constrained to learning a piecewise linear trajectory. Second, we include a proper regularization term to penalize the time warping so it does not ignore parts of the template trajectory. Comparing our results to the Rate Invariant AE shows that this timing regularization is important to combat a degeneracy in the choice of timing of the canonical trajectory. We explain this degeneracy in Section 3.4.1 and explain how it also applies to the work of [98] in the Appendix.

Functional Data Analysis [96] involves the study of how to interpret time-series data, often requiring the registration of data with different timings. The idea of warping the timings of paths appears in, for example, DTW [82] and the Fréchet distance [26]. The Fréchet distance in particular, has good theoretical properties when comparing robot trajectories [39]. One possible issue with the Fréchet distance, however, is that it uses a max operator, returning the distance between the least-well-matched timesteps after optimally aligning the two trajectories. We feel that this might be a problem for our deep learning approach because it throws away gradient information about how well the other parts of the trajectories are aligned, so for our approach, we instead consider the distances between all parts of aligned trajectories rather than just looking at the worst-aligned part of aligned trajectories. It would be interesting to consider some version of the Fréchet distance in future work. This could likely be done with slight modifications to our approach by penalizing only the largest distance between aligned trajectories rather than the distance between all aligned timesteps, as we do. Our work is also related to continuous dynamic time warping [52], which we refine for the manifold-learning setting. The registration and averaging of trajectories is performed in [74], [85], and [99]. Rather than just learning a single average trajectory, we model the full manifold of trajectories. Time-warping is used in [10] to learn "discriminative prototypes" of trajectories, but not a manifold representation of all the trajectories. A linear model to give a learned representation of registered trajectories is generated in [50], and our work can be considered an expansion of that work, using manifold learning to allow for nonlinear spatial variations of registered trajectories.

Time-warping has previously been combined with manifold learning to generate representations of individual frames of a trajectory. For example, [101], [92], and [19] align trajectories and learn representations of individual frames contained in the trajectories. Connectionist Temporal Classification (CTC) can also be viewed as an algorithm for learning a labeling of frames of a trajectory while ignoring timing variations [30]. Instead, our approach focuses on learning a latent vector representation that captures information about the entire trajectory.

Trajectory data can be parameterized in many ways when presented to the

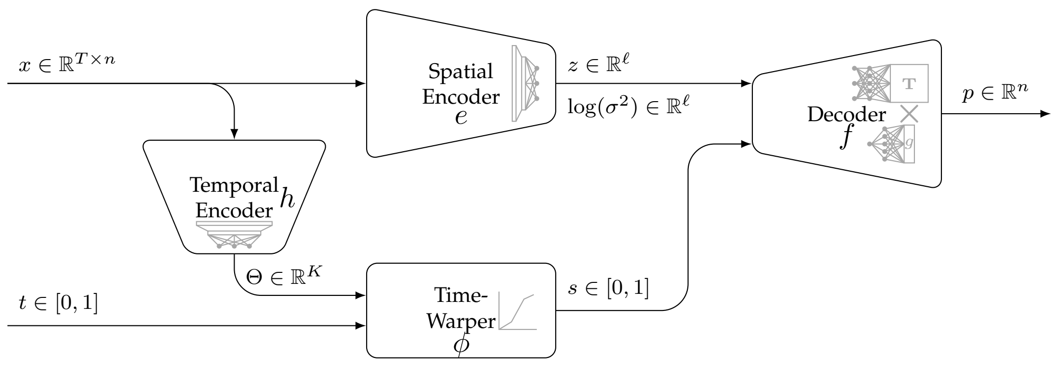

$x \in \mathbb{R}^{T\times n}$
Spatial Encoder e
$z \in \mathbb{R}^{\ell}$
$\log(\sigma^2) \in \mathbb{R}^{\ell}$
Decoder f
T
g
$p \in \mathbb{R}^n$
Temporal Encoder h
$\Theta \in \mathbb{R}^K$
$t \in [0,1]$
Time-Warper ϕ
$s \in [0,1]$

Figure 3.3: The architecture for TimewarpVAE. TimewarpVAE takes in a full trajectory z and a timestamp t and reconstructs the position p of the trajectory at that timestamp. TimewarpVAE separately encodes the timing of the trajectory into Θ and encodes the spatial information into a latent distribution parameterized by z and $\log(\sigma^2)$.

learning algorithm. For example, the trajectory could be parameterized as a dynamic movement primitive (DMP) [41] before learning a (linear) model of DMP parameters, as is done in [62]. A DMP can also be used to learn a representation of states [13] and to model trajectories; for example, the timing can be slowed during execution to allow a robot to "catch up" and correct for execution errors [83]. However, that work does not model timing variations during training. We find the Parametric DMP model is not as accurate as TimewarpVAE. TimewarpVAE accounts for timing variations during training, enabling its latent variable to concentrate its modeling capacity on spatial variations of trajectories.

3.3 TimewarpVAE Technical Approach

A standard approach to learning a manifold for trajectories (see, for example, the method proposed by [11]) is to map each trajectory to a learned representation that includes information about timing and spatial variations. This type of representation learning can be performed by a beta-VAE [35]. TimewarpVAE is based on beta-VAE, with the goal of separating latent variables for spatial and timing factors to make the models more useful for robot execution. To separate the spatial and temporal variations in trajectories, TimewarpVAE contains two modules not present in beta-VAE: a temporal encoder and a time-warper. The decoder now takes in information from both the spatial encoder and the time-warper. Fig. 3.3 shows the architecture of TimewarpVAE. It takes in two inputs: the training trajectory x and a desired reconstruction time t. Like in beta-VAE, the spatial encoder maps the trajectory x to its latent value distribution parameterized by z and $\log(\sigma^2)$. The temporal encoder computes time-warping parameters Θ, and the time-warper (defined by the parameters Θ) now acts on t to warp it to a "canonical time" s. The decoder takes the canonical time s and the spatial latent vector and returns the position of the canonical trajectory for that latent vector at the canonical time s. These modules are further explained in Section 3.4. These functions are jointly trained so the decoded position is a good reconstruction of the position of trajectory x at timestep t, while at the same time minimizing the information that we allow the autoencoder to store about the trajectory.

Specifically, the minimization objective for TimewarpVAE, denoted $\mathcal{L}$, is the sum of the reconstruction cost $\mathcal{L}_R$, beta-VAE's KL divergence loss $\mathcal{L}_{\text{KL}}$, and a new time-warping regularization $\mathcal{L}_\phi$, which we explain further in Section 3.4.1. $\mathcal{L} = \mathcal{L}_R + \mathcal{L}_{\text{KL}} + \mathcal{L}_\phi$, where

$$\mathcal{L}_R = \frac{1}{\sigma_R^2}\mathbb{E}_{x_i,t,\epsilon}\left[\left\|x_i(t) - f\left(\sum_{j=1}^{k} h(x_i)_j\psi_j(t),\, e(x_i)+\epsilon\right)\right\|^2\right] \tag{3.1}$$

$$\mathcal{L}_{\text{KL}} = \beta \mathbb{E}_{x_i} \left[\frac{\|e(x_i)\|^2 + \sum_d (\sigma_d^2(x_i) + \log(\sigma_d^2(x_i))}{2} \right] \tag{3.2}$$

$$\mathcal{L}_{\phi} = \lambda \mathbb{E}_{x_i} \left[\frac{1}{K} \sum_{j=1}^{K} (h(x_i)_j - 1) \log(h(x_i)_j) \right] \tag{3.3}$$

For clarity, we again use the subscript x_i to emphasize that these losses are computed as empirical expectations over each training trajectory x_i. e, σ^2 (and its elements σ_d^2), ϵ, and σ_R^2 are all defined in the same way as for beta-VAE in Eqs. 1.2 and 1.3. f is the decoder, now taking in a canonical timestamp along with the latent value. h is the temporal encoder, so that $h(x_i)_j$ is the jth output neuron (out of K total) of the temporal encoder applied to the ith trajectory. The ψ_j are the time-warping basis functions, explained in Sec. 3.4 and defined in Equation 3.4. β is a regularization hyperparameter for beta-VAEs. λ is a regularization hyperparameter for our time-warping functions, which we set to 0.05 in our experiments. Since λ penalizes the time-warping in the model, a large $\lambda \to \infty$ would drive time-warping to be the identity function, and $\lambda = 0$ could allow the model to learn severe time-warps.

For training, we take a batch of 64 trajectories and compute the spatial latent vector, spatial noise parameter vector, and timing latent vector for each trajectory. A randomly sampled noise term is added to each of the 64 spatial latent vectors according to its associated spatial noise parameter vector. A set of new warped timesteps are computed using the temporal latent vectors. Finally, we decode the full decoded trajectory for each input trajectory by pairing the noisy spatial latent with its associated warped timesteps and decoding repeatedly. This can be done efficiently by repeating each of the 64 spatial noisy latent vectors 200 times (one for each warped timestep), pairing each repetition with a warped timestamp, and then passing all those pairs of latent values and warped timestamps to the decoder in a large batch of $64 \times 200 = 12,800$ values to decode into positions. Finally, we compare each reconstructed position with the input trajecory position and backpropagate the complete loss according to the equations above in a minibatch and backpropagate all the networks using this loss function. In total, we train for 20,000 epochs. We explain the neural network implementation of all of these functions in the next section.

The benefits of this algorithm are as follows: it learns a low-dimensional representation of spatial variations in trajectories; it can be implemented as a neural network and trained using backpropagation; and it can accommodate nonlinear timing differences in the training trajectories. Additionally, new trajectories can be generated using a latent value z and canonical timestamps s ranging from 0 to 1 without using the time-warper or the temporal encoder, which we do in our empirical evaluations, calling these generated trajectories “canonical” or “template” trajectories. These trajectories outperform baseline trajectories qualitatively in Fig. 3.2e and quantitatively in our results section.

3.4 TimewarpVAE Neural Network Formulation

This section explains how to write the spatial encoding function, the temporal encoding function, the time-warping function, and the decoding function as differentiable functions. The time-warping function is a differentiable function with no learnable parameters since the time-warping is entirely defined by the input parameters Θ. The other three modules have learnable parameters, which we learn through backpropagation.

Architecture for the time-warper. The time-warper takes in a training timestamp t for a particular trajectory and maps it monotonically to a canonical timestamp s. ϕ is a piecewise linear function of t with equally spaced knots and K linear segments. The slopes of those segments are labeled by Θ_j for $1 \leq j \leq K$. Different vector $\Theta \in \mathbb{R}^K$ choices give different time-warping functions. In order for Θ to yield a valid time-warping function mapping $[0, 1]$ to $[0, 1]$, the Θ_j should be positive and average to 1. These Θ values are generated by the temporal encoding function discussed in the next paragraph. Given some vector Θ, corresponding to the slope of each segment, the time-warper ϕ is given by $\phi(t) = \sum_{j=1}^{K} \Theta_j \psi_j(t)$ where the ψ_j are defined as follows and do not need to be learned:

$$\psi_j(t) = \min \Big\{ \max \big\{ t - (j-1)/K, 0 \big\}, 1/K \Big\} \tag{3.4}$$

A visualization of these basis functions ψ_j is presented in Fig. 3.4. We use the specific parameterization of Θ_j described in the next paragraph to ensure that our time-warping function is a bijection from $[0, 1]$ to $[0, 1]$. Given an input t, the time-warper is implemented by first computing the $\psi_j(t)$ as a vector (with one element for each j) and then taking the dot product of that vector with the Θ_j's. In our experiments, we set $K = 50$.

Neural network architecture for the temporal encoder. A neural network $h : \mathbb{R}^{n \times T} \to \mathbb{R}^K$ computes a different vector Θ for each training trajectory x. To ensure the elements of Θ are positive and average to 1, the softmax function is applied to the last layer of the temporal encoder, and the result is scaled by K.

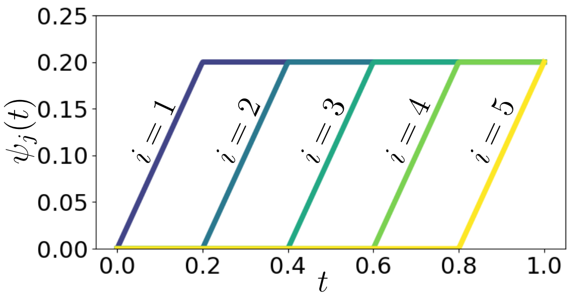
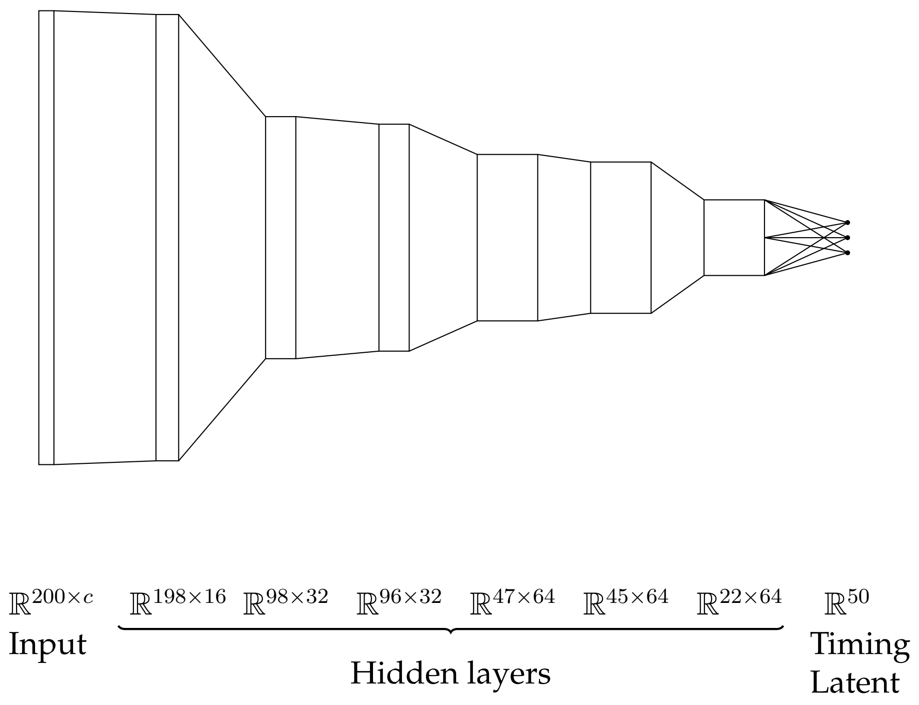


Figure 3.4: Time alignment basis functions for $K = 5$, for each i from 1 to 5. In our experiments, we use $K = 50$

This transformation sends the values θ to $\Theta_j = \mathrm{Softmax}\left(\theta\right)_j K$ for j from 1 to K, ensuring that the average of the output neurons Θ_j is 1 as desired. By contrast, [51] square the last layer and then normalize. The specific architecture used in our experiments, with the shapes of the computed hidden layers, is shown in Fig. 3.5.

Neural network architecture for the spatial encoder. Given a trajectory x evenly sampled at T different timesteps t_j between 0 and 1, the $T \times n$ matrix of these evenly sampled positions $x(t_j)$ is written as $x \in \mathbb{R}^{n \times T}$. In the neural network architecture used in our experiments, one-dimensional convolutions are applied over the time dimension, treating the n spatial dimensions as input channels. This is followed by different fully connected layers for e and for $\log(\sigma)$. However, any neural network architecture, such as a Transformer [94] or Recurrent Neural Network [36], could be used in the spatial encoder module of a TimewarpVAE. Recurrent Neural Networks have shown some benefit for this type of problem, as shown in [32], and should be further explored in future work. The specific architecture used in our experiments, with the shapes of the

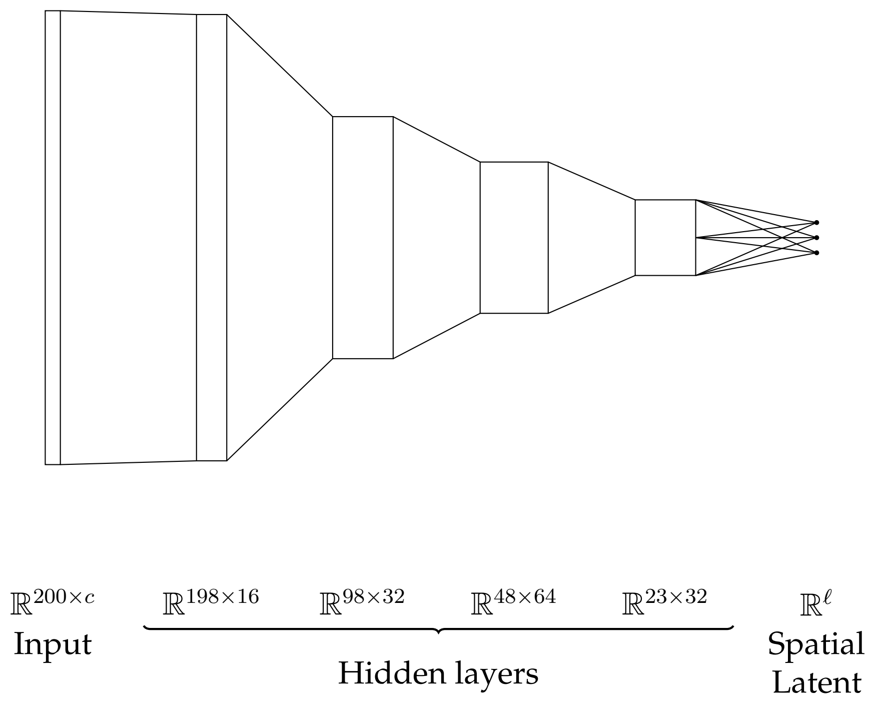


$\mathbb{R}^{200\times c}$
$\mathbb{R}^{198\times 16}$
$\mathbb{R}^{98\times 32}$
$\mathbb{R}^{96\times 32}$
$\mathbb{R}^{47\times 64}$
$\mathbb{R}^{45\times 64}$
$\mathbb{R}^{22\times 64}$
$\mathbb{R}^{50}$
Input
Hidden layers
Timing Latent

Figure 3.5: The temporal encoder architecture includes a series of 1D convolutional steps with kernel size 3 (convolving over the time dimension) that takes in a trajectory with 200 timesteps and c channels. For the handwriting dataset, $c = 2$, corresponding to the x and y positions. For the fork dataset, $c = 7$, corresponding to the fork pose. These convolutions generate hidden layers with data shapes shown in this figure. The last convolution returns an output with 22 timesteps and 64 channels. This last hidden layer is then flattened and fed through a single fully connected layer to compute the spatial latent value estimate, and through a different fully connected layer with the same size to compute the diagonal covariance estimate of the spatial latent value (not shown) each of dimension ℓ equal to the selected latent dimension size.

computed hidden layers, is shown in Fig. 3.6.

Neural network architecture for the decoder Any architecture that takes in a time s and a latent vector z and returns a position p could be used for the decoder f. Our experiments use a modular decomposition of f, with $f(s, z)$ decomposed as the product of a matrix and a vector: $f(s, z) = \mathbf{T}(z)g(s)$. In this



$\mathbb{R}^{200\times c}$
Input
$\mathbb{R}^{198\times 16}$
$\mathbb{R}^{98\times 32}$
$\mathbb{R}^{48\times 64}$
$\mathbb{R}^{23\times 32}$
Hidden layers
$\mathbb{R}^{\ell}$
Spatial Latent

Figure 3.6: The spatial encoder we use in our experiments includes a series of 1D convolutional steps with kernel size 3 (convolving over the time dimension) that take in a trajectory with 200 timesteps and c channels. For the handwriting dataset, $c = 2$, corresponding to the x and y positions. For the fork dataset, $c = 7$, corresponding to the fork pose. These convolutions generate hidden layers with data shapes shown in this figure. The last convolution returns an output with 23 timesteps and 32 channels. This last hidden layer is then flattened and fed through a single fully connected layer to compute the spatial latent value estimate, and through a different fully connected layer with the same size to compute the diagonal covariance estimate of the spatial latent value (not shown) each of dimension ℓ equal to the selected latent dimension size.

formulation, the matrix $\mathbf{T}(z)$ is a function of the latent vector z, and the vector $g(s)$ is a function of the (retimed) timestep s. If each point in the training trajectory has dimension n, and for some hyperparameter m for our architecture, the matrix $\mathbf{T}(z)$ will have shape $n \times m$, and the vector $g(s)$ will be of length m. The nm elements of $\mathbf{T}(z)$ are the (reshaped) output of a sequence of fully connected layers taking the vector z as an input. The m elements of $g(s)$ are computed as the output of a sequence of fully connected layers taking the scalar s as an input. Because the scalar s will lie in the range $[0, 1]$, it is useful to customize the initialization of the weights in the first hidden layer of $g(s)$. Details of the custom initialization are provided in the Appendix. This chosen architecture is useful for easily creating the "NoNonlinearity" ablation experiment in Section 3.5.5. With this architecture, if the hidden layers in $\mathbf{T}(z)$ are removed, then it a linear function of z, and so then entire decoder becomes linear with respect to z, so the generated position p will be a linear combination of possible positions at timestep s. The specific architecture used in our experiments, with the shapes of the computed hidden layers, is shown in Fig. 3.7.

3.4.1 Regularization of Time-Warping Function

The choice of timing for the canonical trajectories adds a degeneracy to the solution.[3] Without our regularization, it is possible for other methods, like Rate Invariant AE, to warp so severely that they can learn to ignore parts of the canonical trajectory. This is a problem if we generate robot motions from the canonical trajectory since it can lead to jittering motions, as seen at the beginning and end of the learned trajectory in Fig. 3.2d.

We propose a regularization penalty on the time-warper ϕ to choose among the degenerate solutions. The proposed penalty is on $\int_0^1 (\phi'(t) - 1) \log(\phi'(t))\, dt$. That regularization contains the function $g(x) = (x - 1)\log(x)$ applied to the slope $\phi'(t)$, integrated over each point t. That function $g(x)$ is concave-up for

[3]This is similar to the degeneracy noted in the Appendix for continuous dynamic time warping [52], and, as noted in the Appendix, was not properly analyzed in [98].

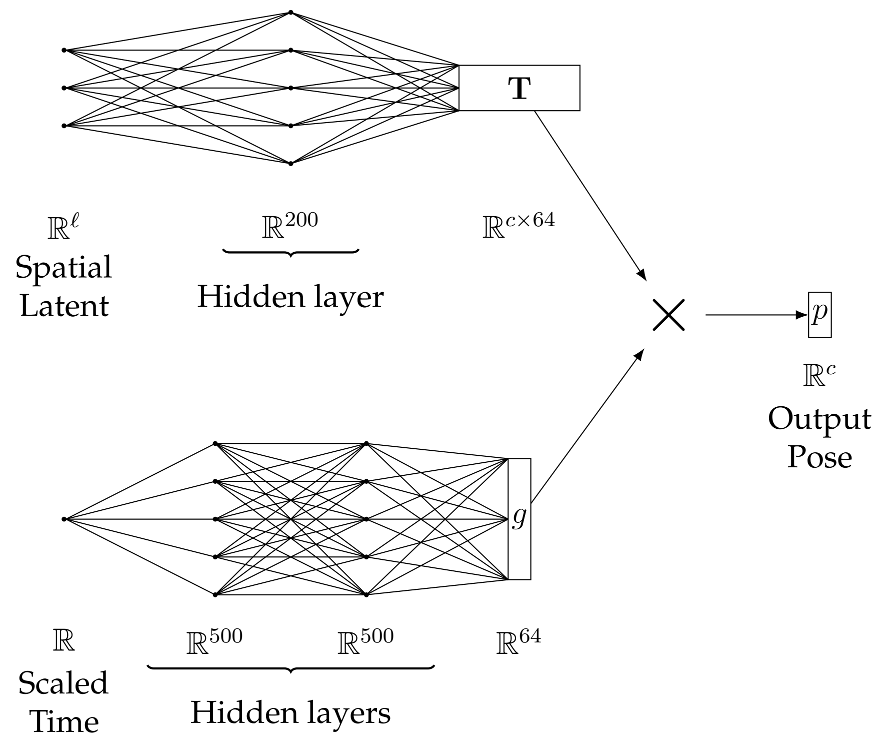
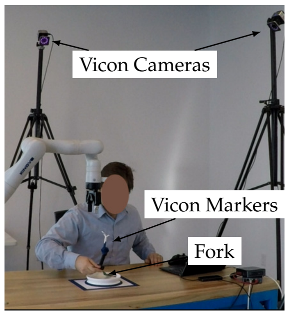
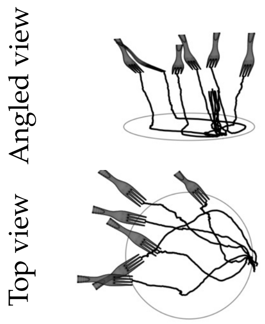


Figure 3.7: The decoder architecture contains fully connected layers and matrix multiplication. One set of fully connected layers (top) takes the spatial latent value as input and computes a single hidden layer of size 200 and then computes a (reshaped) output of size $c \times 64$, labelled **T**. For the handwriting dataset, $c = 2$, corresponding to the x and y positions. For the fork dataset, $c = 7$ corresponding to the fork pose. The other set of fully connected layers (bottom) takes in the scaled time as a single input and computes two sequential hidden layers of size 500 and then computes a vector output of size 64, labelled g. Finally, the matrix-vector product $\mathbb{T}g$ returns the computed pose p (of dimension c). This is the generated pose for to the given spatial latent value and scaled timestep. This process is repeated for different timesteps to generate the full generated trajectory for a particular latent value.



(a) The demonstration system



(b) Six demonstrations[4]

Figure 3.8: We collect trajectory recordings of the position and orientation of a fork while it is used to pick a small piece of yarn off a plate with steep sides. Example trajectories are presented from two angles, showing the initial orientation of the fork and the position of the tip of the fork over time.

$x > 0$ and takes on a minimum at $x = 1$, encouraging the slope of ϕ to be near 1. This formulation has the symmetric property that regularizing $\phi(t)$ gives the exact same regularization as regularizing $\phi^{-1}(s)$. This symmetry is proven in the Appendix. For each trajectory x_i our time-warper ϕ is stepwise linear with K equally-sized segments with slopes $\Theta_1 = h(x_i)_1, \ldots, \Theta_k = h(x_i)_K$. Thus, the regularization integral for the time-warper associated with x_i is

$$\mathcal{L}_\phi(x_i) = \frac{1}{K}\sum_{j=1}^{K}(h(x_i)_j - 1)\log(h(x_i)_j) \tag{3.5}$$

3.5 TimewarpVAE Experiments

Experiments are performed on two datasets, one containing handwriting gestures made in the air while holding a Wii remote [12], and one that we collect ourselves, containing motions that pick up yarn off a plate with a fork, mimicking food acquisition. The same model architecture is used for both experiments, shown in Fig. 3.7 and with additional hyperparameter information given in the Appendix. Additionally, during training, we perform data augmentation by randomly perturbing the timesteps used when sampling the trajectory x, using linear interpolation to compute sampled positions. Our specific data-augmentation implementation is described in the Appendix. This data augmentation decreases training performance but greatly improves test performance, as shown in the ablation studies in Section 3.5.5.

3.5.1 Fork Trajectory Dataset

345 fork trajectories are recorded using the Vicon tracking system shown in Fig. 3.8a. Reflective markers are rigidly attached to a plastic fork, and the trajectory of the fork is recorded using Vicon cameras. A six-centimeter length of yarn is placed on the plate in an arbitrary location and orientation. It is then picked up right-handed by scraping it to the left and using the side of the plate as a static tool to push it onto the fork. Demonstrations were intentionally collected with three different timings, where in some trajectories the approach to the plate was intentionally much faster than the retreat from the plate, in some trajectories the approach was intentionally much slower than the retreat from the plate, and in the remaining trajectories the approach and retreat are approximately the same speed. The dataset was split into 240 training trajectories and 105 test trajectories. Examples of six recorded trajectories, along with visualizations of the starting pose of the fork for those trajectories, are presented in

[4]Fork meshes in 3D trajectory plots were downloaded and modified from https://www.turbosquid.com/3d-models/metal-fork-3d-model/362158 and are used under the TurboSquid 3D Model License

Fig. 3.8b. Trajectories are truncated to start when the fork tip passes within 10cm of the table and to end again when the fork passes above 10cm. All trajectories were subsampled to 200 equally-spaced time points, using linear interpolation as needed. We express the data as the x, y, z position of the tip of the fork and the rw, rx, ry, rz quaternion orientation of the fork, giving a dataset of dimension $n = 7$ at each datapoint. The data is preprocessed to choose a consistent sign of the quaternion representations so they are all near each other in $\mathbb{R}^4$. The data is mean-centered by subtracting the average x, y, z, rw, rx, ry, rz training values, and the x, y, z values are divided by a normalizing factor so that their combined variance $\mathbb{E}[x^2 + y^2 + z^2]$ is 3 on the training set. Likewise, the rw, rx, ry, rz values are multiplied by a scaling factor of $0.08m$, (chosen as a length scale associated with the size of the fork), before dividing by the same normalizing factor, to bring all the dimensions into the same range.

3.5.2 Handwriting Gestures Dataset

The air-handwriting dataset collected by [12] is used for the handwriting experiment. The drawn letters are projected onto xy coordinates. A training set of 125 random examples of the letter "A" is drawn from the air-handwriting dataset, and the remaining 125 random examples of the letter "A" are the test set. All trajectories were subsampled to 200 equally-spaced time points, using linear interpolation as needed. The data are mean-centered so that the average position over the whole training dataset is the origin, and x and y are scaled together by the same constant so that their combined variance $\mathbb{E}[x^2 + y^2]$ is 2 on the training set. Example training trajectories of handwritten letter "A"'s are shown in

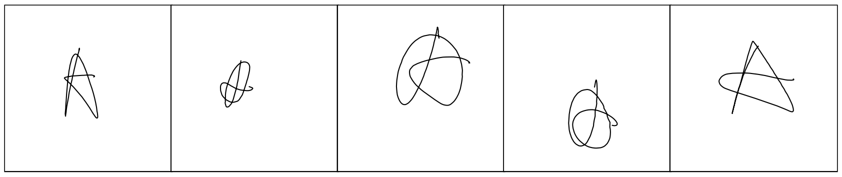


Figure 3.9: Five example trajectories of handwritten "A"

Architecture	Beta	Rate	Training A-RMSE ($\pm 3\sigma$)	Test A-RMSE ($\pm 3\sigma$)
TimewarpVAE	0.01	3.716	0.187 ± 0.003	$\mathbf{0.233 \pm 0.003}$
	0.1	3.227	$\mathbf{0.185 \pm 0.007}$	0.234 ± 0.008
RateInvariantAE	0.01	4.095	0.260 ± 0.130	0.316 ± 0.188
	0.1	3.280	0.285 ± 0.154	0.325 ± 0.132
beta-VAE	0.01	4.759	0.291 ± 0.005	0.343 ± 0.016
	0.1	3.670	0.293 ± 0.007	0.342 ± 0.011
NoTimewarp	0.01	3.924	0.264 ± 0.007	0.360 ± 0.017
	0.1	3.508	0.265 ± 0.006	0.354 ± 0.014

Table 3.1: Performance results for 3-dimensional models of fork trajectories. Our TimewarpVAE significantly outperforms beta-VAE and the ablation of TimewarpVAE without the time-warper.

Fig. 3.9.

3.5.3 Model Performance Measures

The performance measures include three important values: the training reconstruction error, the test reconstruction error, and the model rate. Since we are interested in the ability of our model to measure spatial variation of trajectories, reconstruction errors are computed by first performing symmetric DTW to align the reconstructed trajectory to the original trajectory. We then compute the Euclidean mean squared error between each point in the original trajectory and every point it is paired with. After that, we calculate the average of all those errors over all the timesteps in the original trajectory before taking the square

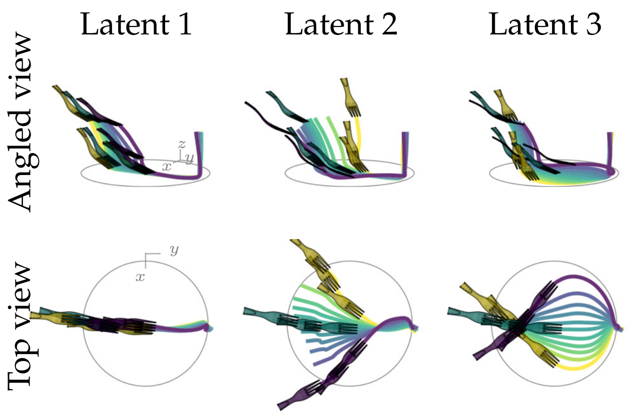


Figure 3.10: The paths of the fork tip are plotted over time for TimewarpVAE trajectories using different latent vectors. The orientation of the fork is shown at three different timesteps in a color matching the associated path. Each latent dimension has an interpretable effect. The first latent determines the fork's initial y position, the second latent determines the fork's initial x position, and the third latent determines how the fork curves during trajectory execution.

root to compute our aligned root mean squared error (A-RMSE). In the framework of Rate-Distortion theory, these errors are distortion terms. The model rate measures the information bottleneck imposed by the model and is given by the KL divergence term in the beta-VAE loss function. It's important to check the rate of the model since arbitrarily complex models can get perfect reconstruction error if they have a large enough rate [3]. However, among trained models with similar rates, it is fair to say that the model with lower distortion is a better model. Model rate is reported in bits.

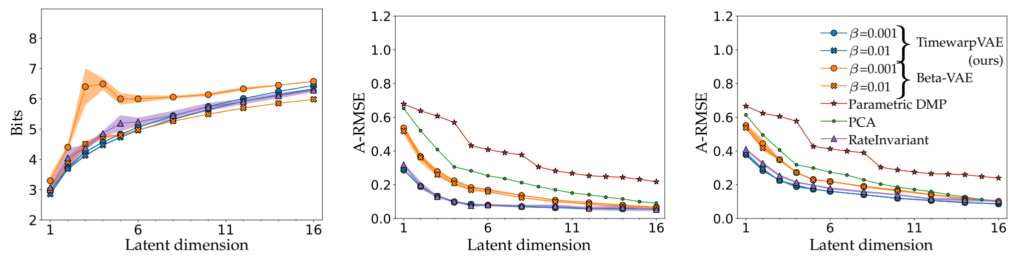
3.5.4 Fork Model Results

Models are trained with a latent dimension of three on the fork dataset. TimewarpVAE is compared to beta-VAE, a VAE version of Rate Invariant AE, and an ablation of TimewarpVAE called "NoTimewarp" that sets the time-warping module to the identity function. The latent sweep of a TimewarpVAE trained with $\beta = 1$ is shown in Fig. 3.10, showing its interpretable latent space. Table 3.1 shows performance measures, where we compute and summarize five trials for each model type for various hyperparameters β. TimewarpVAE outperforms the baseline methods.

3.5.5 Air Handwriting Results

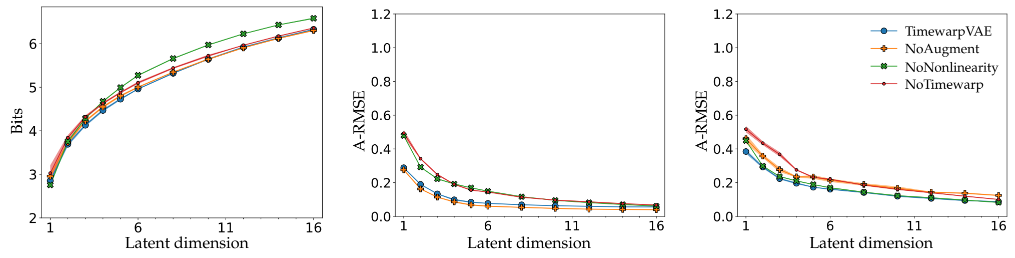
TimewarpVAE is compared to baseline methods trained on the same training and test splits. All models are trained with a batch size of 64 for 20,000 epochs, using the Adam optimizer and a learning rate of 0.001. We train each model five times for different choices of beta, and we plot the mean and one standard error above and below the mean. To give context to these results, we also show results for parametric dynamic movement primitives (Parametric DMPs) [62] and PCA results, which do not have associated rate values. TimewarpVAE significantly outperforms Parametric DMPs, PCA, and beta-VAE, with the greatest differences for smaller latent dimensions. TimewarpVAE shows comparable performance to RateInvariant on training error and consistently outperforms RateInvariant in test error.

Ablation studies were run to understand the importance of different architecture choices. The first removes the data augmentation of additional trajectories with perturbed timings. We call this ablation"NoAugment." The second removes the hidden layers in the neural network $\mathbf{T}(z)$. Removing the hidden lay-



(a) Rate (b) Train Distortion (c) Test Distortion

Figure 3.11: TimewarpVAE compared to beta-VAE, Parametric DMP, PCA, and Rate Invariant AE. Especially for lower-dimensional models, TimewarpVAE performs comparably to RateInvariant on training error and consistently outperforms RateInvariant in test error.



(a) Rate (b) Train Distortion (c) Test Distortion

Figure 3.12: Ablation results show the data augmentation of timing noise is important for generalization performance, the nonlinear model gives a better fit to training data without losing generalization performance, and the time-warper is key to TimewarpVAE's good performance.

ers makes $\mathbf{T}(z)$ a linear function, meaning the function f can only learn trajectories that are a linear function of the latent variable z (but f is still nonlinear in the time argument s). We call this ablation "NoNonlinearity". The third removes the time-warper and replaces it with the identity function. We call this ablation "NoTimewarp." Results for these ablations are plotted in Fig. 3.12. NoAugment confirms the importance of data augmentation for good generalization. NoAugment can get a slightly better fit to the training data than TimewarpVAE, but performs poorly on test data. NoNonlinearity has comparable performance on the test data to TimewarpVAE, showing the strict regularization imposed by its linearity condition is good for generalization. However, NoNonlineary has such strong regularization that it cannot fit the training data as closely. Additionally, the model rate for NoNonlinearity is higher. By constraining our model to be linear in the latent term, we cannot compress information as efficiently into the latent space. Without the ability to time-align the data, NoTimewarping is not able to fit either the training or test data well. The information rate of NoTimewarping is the same as that of TimewarpVAE, showing it does not compress spatial information as efficiently.

When reading these results, it's important to note that while adding more latent dimensions is an easy way to improve the model performance, when actually using the generative model we want the latent space to be easy to search to select a trajectory, so an actual use case would favor smaller latent spaces that capture the important factors of variation of the dataset and don't additionally allocate latent dimensions to modeling small variations in the data. The choice of the right latent space size to use would depend on the particular dataset and use case of the learned model. Additionally, hyperparameters were chosen to reduce holdout error. The large modeling capacity of the deep learning methods for large latent dimensions would make it very easy to do much better than PCA on the training dataset if we wanted to, but for the small training datasets we use, such models would overfit and do very poorly on the holdout dataset. There is a limit to the holdout performance that can be robustly attained on a small training dataset, based on the intrinisic noise in the data.

3.6 Letter 'A' Robot Execution

We demonstrate the usefulness of our algorithm on a Kinova Gen3 robot arm. Because of our timing regularization term, we can optimize various timing options during the replay of the trajectory, constraining the trajectory to stay close to the demonstrated timings. This timing optimization allows the robot to consider its joint torque and speed limitations and choose a timing that it can execute the quickest under those constraints. An example is shown in Fig. 3.1. If the robot can only scale the timing uniformly, the optimized trajectory takes 1.8 times longer to execute. Our approach gives the robot the flexibility to slow down some parts of the trajectory and speed up other parts.

3.7 Picking Up Yarn Simulated and Physical Experiments

We additionally ran simulated and physical experiments, picking up a short yarn segment using a fork. For these experiments we happened to also use a CurvVAE regularization and a slightly different model architecture, as explained below.

Picking Up Yarn Baseline Method Definition

When presented with a new object configuration, the simplest way to use the human demonstration data would be to replay the demonstration trajectory that had an initial object configuration that was most similar to the new con-

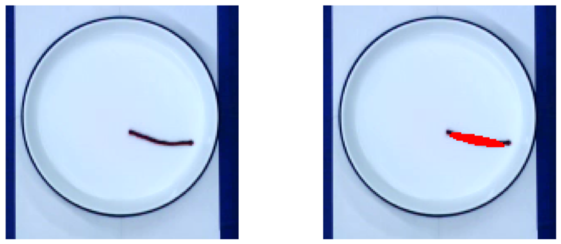


Figure 3.13: Robot camera image before and after string localization

figuration. We label this approach 1NN (one nearest neighbor) and use this as a baseline approach.

In order to implement the nearest-neighbor algorithm, during trajectory collection we additionally need to record where the string was for each demonstration. To identify the object configuration on the plate during demonstration collection, we use the wrist-mounted RGB camera on the Kinova Gen3 arm to take an overhead picture of the scene, with the robot taking the photograph using the pose shown in Figure 3.8a We use color segmentation to identify the pixels corresponding to the target object and model the object configuration using a two-dimensional Gaussian distribution, computing the mean pixel location and the sine and cosine of twice the orientation angle of the major axis of variation. We use sine and cosine so that an angle of 0 and 2π are correctly identified, and we use twice the orientation angle so that a rotation of the string around its center by π radians will also be seen as a similar configuration. We use these four features in the nearest neighbor calculation to compute which training trajectory to execute. An example showing the overhead image of the plate and the detected object configuration is shown in Figure 3.13.

Picking Up Yarn Model Details

As before, in this experiment the latent space is three-dimensional and each pose in a trajectory as a seven-dimensional vector containing the position of the tip of the fork and the quaternion representing its orientation. For this experiment, we only subsampled to 100 different waypoints to represent each trajectory. The quaternion elements were scaled by an additional factor that they were of comparable size to the position values which are in units of meters. For this experiment we happened to use a length scale of 0.04m, which is roughly the length of the prongs of the fork, rather than the 0.08m length scale in the previous model.

Other slightly different hyperparameters were $k = 20$ basis functions in the time alignment module (rather than the 50 tested above). The architecture was much simpler, using two 1D convolutional layers followed by two fully-connected layers for the embedding functions in our VAE and also for modeling the time-warping parameters Θ. For our canonical motion module that takes in a latent and a time and returns the pose, we just used two fully-connected hidden layers. Additionally, for these experiments we added a CurvVAE penalty to the spatial encoding as well. These differences suggest that our approach can work for a variety of architectures.

We performed an exploratory search for hyperparameters that gave good performance on the holdout set, and we tuned the γ parameter of the CurvVAE to just slightly begin to degrade test performance. Our final hyperparameter choices were to set the β-VAE $\beta = 0.0001$, the learning rate for the VAE parameters to be 0.0001, the learning rate for the time transformation module to be

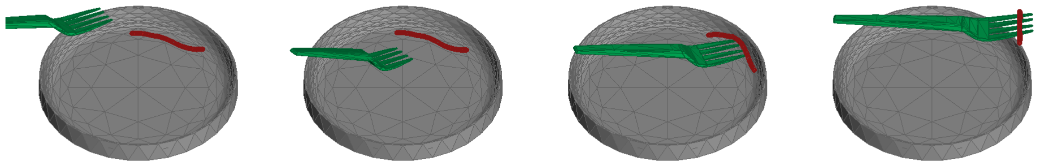


Figure 3.14: Images from an example simulation run

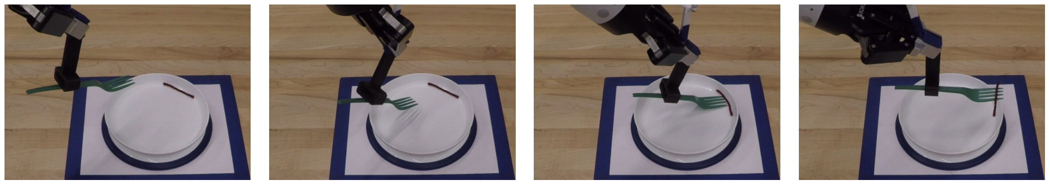


Figure 3.15: Images from an example physical robot run

0.001, and the γ for the CurvVAE penalty to be 0.00001.

We train for 50,000 epochs on a Quadro GV100 GPU, which takes approximately 2 hours. The resulting model was very similar to the previously trained model, and for completeness, we present a visualization of latent sweeps in the Appendix.

Picking Up Yarn Trajectory Search in Simulation

We use PhysX [66] to simulate the fork trajectory's effect on the environment. We measured the contours of an OXO Tot Stick & Stay Suction Plate, and created a triangle mesh to represent it. We ran physical experiments using an EcoChoice Heavy Weight 6 1/2" Green CPLA Plastic Fork, and we were able to find a 3D model of that fork from the seller which we decomposed and simplified into convex hulls and used in our simulation[5]. We modeled the yarn as a chain of eleven overlapping capsules, with total fully outstretched length of 7cm. The connections between capsules have stiffness to them, preferring to return to a pre-specified resting angle. We add noise to generate non-zero resting angles between the capsules, decreasing the resting length of the chain and adding stochasticity to the simulation. Still images from a simulation run are shown in Fig. 3.14.

We do not use any of the string location data when building our unsupervised models of trajectory. However, to speed up the search, we seed our latent-space search with the embedding of the 1NN baseline trajectory.

Picking Up Yarn Physical Robot Execution

We replay the trajectory using a greedy Jacobian-based controller to move the end-effector along the desired path. We use the average timing across training data when replaying waypoints in trajectories, slowed down for execution on the robot. We use rejection sampling to filter starting joint angles so that the full trajectory is feasible, avoids self-collisions and collisions with the environment, and avoids joint singularities. Avoiding joint singularities is important because, for safety, we regularize the calculation of the required joint angle changes to track the desired poses. Thus, when the robot is near singularities it will not follow the target trajectory as accurately.

Additionally, we noticed up to an approximate 14mm max tracking error

[5]Fork purchased from https://www.webstaurantstore.com and 3D model downloaded, modified, and used under Fair Use

between the calculated kinematics of the Kinova Gen3 robot arm and the actual position of the end-effector measured using a Vicon system. The error direction varied depending on the configuration of the robot arm. To decrease this tracking error, during trajectory replay we use feedback from the Vicon system to adjust the target tracking position of the fork tip to adjust for errors, reducing our max tracking error to around 2.6mm. Still images from a physical robot run are shown in Fig. 3.15.

Picking Up Yarn Results

We run 20 trials in simulation for each of nine specified string poses and look at how frequently the yarn ends the episode more than 40mm above the plate. We find the baseline nearest neighbor method is successful 53% of the time, while our proposed algorithm increases the success rate to 71%.

On the physical robot, we run three trials at each of those nine string poses and find that there is a sim-to-real gap. The baseline method is only successful in 26% of the trials, while our method on the physical robot is successful 52% of the time. For the same starting pose of the string and the same robot trajectory, we find that there is stochasticity in whether the trajectory will be successful due to variations in the internal state of the individual threads in the yarn between runs, leading to different frictional and deformation behavior.

3.8 Discussion on Separating Timing and Spatial Latent Variables

TimewarpVAE is useful for simultaneously learning timing variations and latent factors of spatial variations. Because it separately models timing variations, TimewarpVAE concentrates the modeling capacity of its spatial latent variable on spatial factors of variation. As discussed further in the Appendix, TimewarpVAE can be viewed as a natural extension of continuous DTW. TimewarpVAE uses a parametric model for the time-warping function. A different approach could be to use a non-parametric model like DTW to warp the reconstructed trajectory to the input trajectory and then backpropagate the aligned error. That alternate approach has a much higher computation cost because it requires computing DTW for each trajectory at each learning step and does not re-use the time-warper from previous steps. We consider this approach briefly in the Appendix and leave it to future work to more fully understand the benefits of this alternate implementation and when it should be used or combined with the proposed TimewarpVAE. While this work focused on trajectories parameterized by storing the position at each timestep, we believe our approach could also be applied with some modifications to trajectories which are parameterized by storing the velocity of the trajectory at each point, as is done in [32]. To handle trajectories like that, we would need to take into account the fact that time-warping the trajectory also affects the speed of the trajectory at each point. In particular, we believe we could use the chain rule in situations like that. Our decoder network should return the canonical velocity $\frac{df}{ds}(s)$. Our time-warper would need to return both the warped time $s = \phi(t)$ and also the warping velocity at that time $\frac{ds}{dt} = \phi'(t)$ (which we already compute as part of our parameterization). We could then apply the chain rule to say that the warped velocity $\frac{df}{dt}(t) = f'(\phi(t))\phi'(t)$. This approach would allow us to add to the work of [32] and additionally separate the timing and spatial/velocity latent variables. The benefits of parameterizing the trajectory as a velocity instead of a position should be studied in future work. Additionally, it would be interesting in future work to look at a decoder that parameterizes the canonical trajectory as a form of Recurrent Neural Network, as is done in [32]. Since our time-warping requires varying and irregular sampling intervals of the canonical trajectory, we would likely need to use a Continuous Recurrent Unit [84] to handle the varying sampling intervals. This chapter measured the spatial error of reconstructed training and test trajectories, and showed the TrajectoryVAE does a better job than beta-VAE at compressing spatial information into small latent spaces. We demonstrated our algorithm on a Kinova Gen3 robot arm, showing it can generate very fast robot motions.

CHAPTER 4

JACOBIAN L1 VARIATIONAL AUTO-ENCODER

4.1 Underspecification of the Representation Learning Problem

Unsupervised representation can take in a collection of data from the world and figure out how to organize and find patterns in the data without additional information about how the images were generated. The ideal representation learning algorithm would compress high-dimensional data into a lower-dimensional latent representation that contains relevant information about the ground-truth factors of variation that generated the data.

Inferring a good latent representation from a dataset is a difficult problem and is generally underspecified in algorithms. This underspecification is called the "model identification" problem, and one example is that representation learning algorithms often struggle to specify the correct orientation of a latent space. That is, optimization criteria used to learn a representation function might be equally well satisfied by an equivalent representation that is just a rotation of the latent space by an arbitrary amount. This type of rotation of the latent space was visualized and described in Fig. 1.4 and Fig. 1.5.

As commonly implemented (using axis-aligned Gaussian posterior distributions), variational autoencoders (VAEs) [48] and their extension, the beta-VAE [35], solve the rotational part of the model identification issue by tending to ensure that the generation function's Jacobian matrix has orthogonal columns (i.e., that the generative Jacobian matrix's right singular values are aligned with the axes of the latent space) [81, 54]. Intuitively, this is because the VAE's stochastic reconstruction cost prefers to budget higher precision (lower embedding noise) in the directions along which the generative Jacobian changes most rapidly. [54] draw the parallel between this preferred orientation and linear principal component analysis (PCA). However, we note that learning algorithms that resolve rotational identification issues through methods related to PCA will still suffer from an identifiability issue related to rotations that mix the directions for which the generative Jacobian matrix has equal singular values. Along these directions, the posterior Gaussian would have approximately equal variance and be rotationally symmetric.

We propose adding an L_1 cost to the generation function's Jacobian matrix as a way to resolve that rotational identifiability issue. Since L_1 cost is not rotation invariant, L_1 regularization creates a preferred latent-space orientation among directions whose singular values are equal. This use of the L_1 norm to choose an orientation is inspired by similar uses in linear models. For example, when using a Laplacian prior in Independent Component Analysis (ICA) [40] and applying it to already-whitened data, ICA rotates the data to minimize the L_1 norm. As shown in [67], that ICA formulation is equivalent to sparse coding using an L_1 cost [68], with the same rotational effect. In Sparse PCA, the L_1 norm encourages sparsity in the loadings/mixing matrix (rather than the principal components/sources as in ICA) [44, 103], similarly encouraging preferred orientations. These techniques for preferring certain orientations using the L_1 norm are all linear model techniques, and we apply them to nonlinear VAEs by regularizing the L_1 norm of the generator Jacobian matrix, thereby encouraging a preferred orientation for our nonlinear models. This chapter will focus its experiments on image data, but we believe the underlying theory should also be useful for trajectory data.

The motivation above suggests that an L_1 norm on the generator Jacobian can address the rotational identifiability issue, and we think the L_1 regularization will encourage useful orientations of the latent space. This belief comes from results on the sparse linear coding of images. In particular, [68] showed how sparse linear coding on natural images generates local receptive fields similar to those discovered in the mammalian visual processing system. These types of localized receptive fields are shown in Figure 4.1. Each image in that figure corresponds to a direction in the latent space and shows how perturbing the latent value in that direction changes the generated image—white pixels mean the image gets brighter there, and black pixels mean the image gets darker. For the ICA basis, we see that perturbations in different latent directions are associated with localized changes to the image, whereas for the PCA basis, the latent directions tend to affect the entire image. Adding an L_1 penalty to our model should encourage sparsity within the generative Jacobian columns of our model, meaning that perturbations of latent values along individual latent directions should modify as few pixels as possible, leading to latent directions that affect the output image in localized regions. Therefore, L_1 generative Jacobian regularization should disentangle representations of different objects in an image. We call the proposed model trained with this additional regularization a Jacobian L_1 regularized variational autoencoder (JL1-VAE).

The use of the generative Jacobian in the JL1-VAE implies a local linear ap-

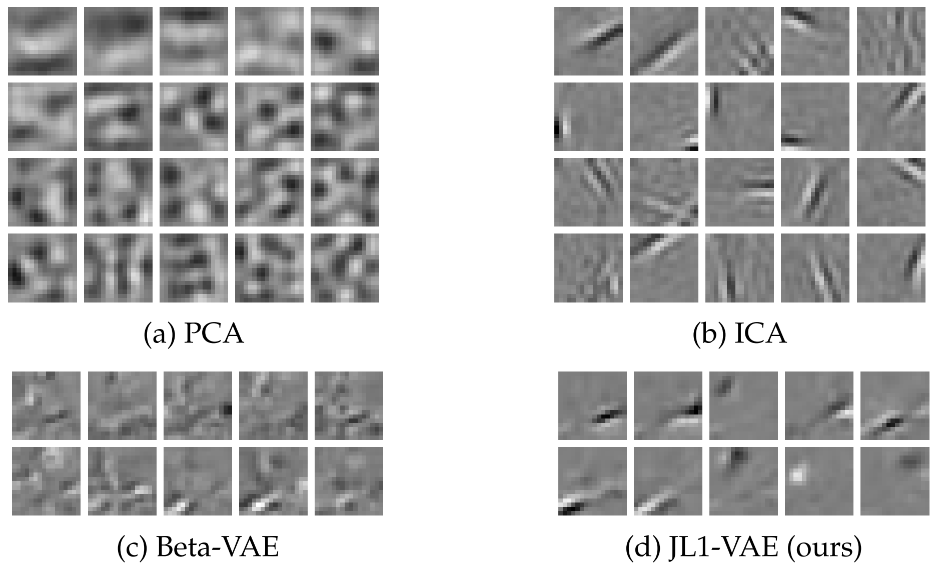


Figure 4.1: Example columns from generative Jacobian matrices for different modeling techniques on natural image data collected in [68]. ICA used 100 latent dimensions, of which 20 samples are shown. Beta-VAE used $\beta = 0.01$. JL1-VAE used $\beta = 0.01$, $\gamma = 0.01$, each on ten latent dimensions.

proximation of the generative function, so, in order for the JL1-VAE to be useful, the training image data should lie on a manifold [86], and should be sufficiently well sampled. Additionally, JL1-VAE contains inductive bias, like every other unsupervised disentanglement algorithm [55]. As mentioned above, JL1-VAE's regularization of the generative Jacobian encourages small changes in latent values to result in sparse (impacting a small number of pixels) changes to the resulting image, similar to local receptive fields. This inductive bias is well suited for disentangling motions of different objects in an image but would presumably not be useful for whole-image changes, such as rotation of the entire image or brightness changes across the whole image.

We apply our novel[1] JL1-VAE framework to a variety of datasets, giving

[1]The work in this chapter was previously published as [76]

qualitative and quantitative results showing that our added L_1 cost can encourage local alignment of the axes of the latent representation with individual factors of variation.

4.2 Motivation for the JL1-VAE Approach

Previous work [54] showed that the stochastic reconstruction term of the ELBO, $E_{z\sim q(z|x)}\left[\log(p(x;g(z)))\right]$, can be approximated using a second-order Taylor expansion as:

$$E_{z\sim q(z|x)}\left[\log(p(x;g(z))\right] \approx \log p(x|h(x)) + \frac{1}{2}\text{tr}\left(J_g(h(x))^\top H_{p_x}(g(h(x)))J_g(h(x))\Sigma_{z|x}(x)\right) \tag{4.1}$$

J_g is the Jacobian of the generator function, and $H_{p_x}(g(h(x)))$ is the Hessian with respect to $g(z)$ of the log of the generative probability $\log(p(x;g(z)))$ evaluated at $g(h(x))$. For standard VAE implementations using diagonal Gaussian posteriors and pixel-factorized generative probabilities, $\Sigma_{z|x}(x)$ and $H_{p_x}(g(h(x)))$ are both diagonal.

Equation 4.1 shows that the stochastic reconstruction loss of the ELBO can be approximated by a deterministic reconstruction loss with a (weighted) L_2 regularization cost on the Jacobian $J_g(h(x))$. [54] use this weighted L_2 regularization in the approximation above to show how the ELBO encourages local alignment of the right singular vectors of J_g to $\Sigma_{z|x}(x)$. That is, larger values of $\Sigma_{z|x}(x)$ (larger embedding noise) in some directions lead to larger L_2 regularization on the generator Jacobian in those directions. Equivalently, in directions with large changes in the generator Jacobian, the regularization encourages smaller embedding noise (more precision) in the posterior $\Sigma_{z|x}(x)$. For this work, we use the presence of an implicit L_2 loss on the generator Jacobian as further motivation for our choice to add an explicit L_1 regularization to the generator Jacobian.

4.3 Related Work on Sparsity and Disentanglement of Latent Representations

There are several areas of related work. Previous foundational work has analyzed the disentanglement properties of beta-VAEs. We delve into previous uses of the term "sparse VAE," as there are (at least) two other common and well-studied meanings of "sparse VAE," which we disambiguate from the type of sparsity we study in this chapter. Additionally, there is previous VAE work inspired by ICA. Different architectural choices have been shown to improve disentanglement. Finally, we discuss modifications to the VAE objective that have previously been studied to improve disentanglement.

Disentanglement of beta-VAEs [61] and [81] show that restricting the posterior covariance to diagonal (called the mean-field assumption) breaks the rotational symmetry of beta-VAEs. [81] further show that this encourages the columns of the generator Jacobian to be orthogonal, relating local beta-VAE latent directions with PCA decomposition. In our work, we take this a step further, explicitly regularizing the sparsity of the generator Jacobian, to break rotational symmetry between directions with equal singular values in the generator Jacobian.

Sparse VAEs: Sparsity in VAE codes Some previous work involving L_1 regularization and VAEs uses the term "sparse VAEs" to refer to sparsity in the *latent values* taken on by the latent codes themselves. These works attempt a minimization of something like $\|z\|_1$. This meaning is studied in [60, 65, 43]. When we use sparsity in the present work, however, we are not concerned with the values of the latent z but rather with how small axis-aligned changes in the latent values affect the output. That is, we are concerned with sparsity of $J_g(z)$, not z. Our associated cost is $\|J_g(z)\|_1$. Our use of "sparsity" concerns local disentanglement rather than sparsity in latent values, which would be a type of global disentanglement.

Sparse VAEs: Sparsity in Network Weights Likewise, other work involving L_1 regularization and VAEs (and neural networks more broadly) uses "sparsity" to refer to the desire to make many network weights 0, with the motivation of reducing the size of the stored neural network architecture. This meaning is seen in [58, 24]. In this work, by contrast, we are interested in sparsity in the generator Jacobian, not in the network weights. We note the distinction between the sparsity of individual network weights and sparsity in the Jacobian of the overall function. Multiplying sparse matrices does not necessarily result in sparse matrices, and nonlinear activations can allow a sparse Jacobian even if the network weights themselves are not particularly sparse. Thus, the $\|J_g(z)\|_1$ cost we study in this work is not what is referred to in studies of the sparsity of neural networks, which consider regularizations like the L_1 cost over network weights.

ICA within the VAE Literature Independent Component Analysis (ICA) [40] is often mentioned in the VAE literature in relation to the role ICA has played in the theory of identifiability and disentanglement of representations [54, 61, 81, 47, 56]. [87] propose using a structured, rotationally asymmetric prior to encourage disentanglement in the embedding. This and other approaches that attempt to match the embedding distribution globally to a desired shape are very different from the local, Jacobian-based approach we take in this paper. [46] relate nonlinear ICA with VAEs in the case where the data has an additional observed variable, and they give a proof that, in that case, their model is identifiable and correctly disentangles the ground-truth factors of variation. We focus on fully unsupervised training data and assume that we are not given access to any data labels. We are not aware of any prior work applying a sparsity cost to the generator Jacobian, which is the inspiration we take from ICA.

Architectures shown to improve disentanglement Previous work has shown impressive results from modifying the network architecture to explicitly represent multiple objects by, for example, learning object masks [31], or by modifying how the latent variable is read into the generative model architecture [97]. [1] and [45] construct model architectures that explicitly encourage sparsity in the generative network weights. In this work, we focus on how we can regularize the objective function to improve disentanglement rather than studying how different network architectures can improve disentanglement.

Modification of VAE prior Previous work has also investigated modifications to the unit Gaussian prior commonly used in VAEs. [91] use a learnable Gaussian mixture prior. [87] use a generalized Gaussian distribution (that is not rotationally invariant) as the prior. [5] use rejection sampling to form a more complicated, non-rotationally-invariant prior. [23] and [71] even modify the prior to lie on non-Euclidean surfaces. [47] do not explicitly enforce a prior distribution but rather use a regularization term to encourage the prior distribution $q(z)$ to be a factorized distribution. While these approaches pressure the entire embedded distribution to have certain properties, we are instead focused on how to modify the learning objective to give *local* bias toward disentanglement rather than using more global methods based on the overall distribution of the embedded dataset.

Regularization of VAEs Several previous works explicitly or implicitly use L_2 normalization of the generator Jacobian [54, 79, 93, 38]. [14] regularize by $\|J_g^\top(z)J_g(z) - c\mathbb{1}\|_2$ for some constant c. We are not aware of previous investigations of $\|J_g(z)\|_1$ regularization for VAEs.

4.4 JL1-VAE Model Loss Calculation and Architecture

4.4.1 Loss Calculation

We define a Jacobian L_1 regularized variational autoencoder (JL1-VAE) as a VAE that is trained using the beta-VAE loss augmented with an L_1 regularization of the Jacobian matrix of the map from latent values to mean generated images. The regularization term is modulated by a hyperparameter γ.

Specifically, the maximization objective for the JL1-VAE is the sum over all datapoints x of

$$L_{\text{JL1}}(x) = E_{z\sim q(z|x)}\left[\log p(x|z) - \gamma\big|\big|J_g(z)\big|\big|_1\right] - \beta\text{KL}\left(q(z|x)\|N(\mathbb{0}, \mathbb{1})\right) \tag{4.2}$$

Since we use a Gaussian posterior $q(z|x) = N\big(h(x), \Sigma_{z|x}(x)\big)$, we can use an explicit calculation for the KL-divergence. We estimate the expectation of $\log p(x|z)$ and of $\gamma\|J_g(z)\|_1$ using a single sample z from the distribution over which we are taking the expectation. We estimate the full Jacobian matrix $J_g(z)$ using the finite difference method along each latent dimension. This leads to a runtime that scales roughly linearly with the number of latent variables in the VAE architecture.

4.4.2 Architecture

We use a convolutional architecture for our VAEs. In particular, our embedding architecture consists of convolutional layers followed by a fully connected layer with ReLU activations. This base model is shared between the mean and log variance embedding networks. Each embedding network then appends its own linear, fully connected head to the shared model. We use a diagonal structure for the log variance estimates to reduce the number of parameters we need to estimate. We use a latent dimension of ten for all experiments, though we have seen similar results for other latent dimension sizes. The reconstruction architecture consists of fully connected layers followed by convolutional layers, using ReLU activations, with a final sigmoidal activation function. The type of architecture used in our experiments is shown in Fig. 4.2. A full set of hyperparameter

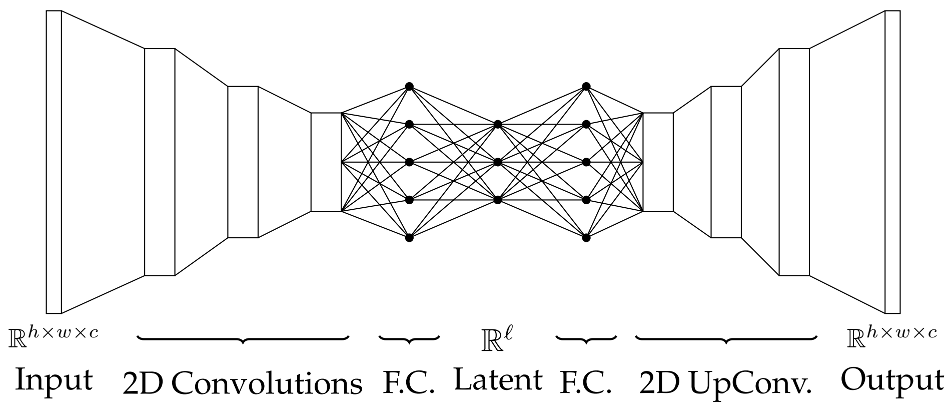


Figure 4.2: The neural network architecture takes in an image (of height h, width w, and channels c) and performs 2D convolutions followed by fully connected layers to compute an associated latent value of dimension ℓ. The decoder is the reverse, consisting of fully connected layers, followed by 2D convolutional transpose layers (upconvolutions), finally returning an image as the last layer's output. Specifics on the number and shape of each type of layer for the different experiments are contained in the Appendix.

choices for each experiment can be found in the Appendix. When we compare JL1-VAE with other methods, we ensure consistent architecture choices.

4.5 JL1-VAE Experiments

4.5.1 Datasets

To evaluate the ability of JL1-VAE to disentangle factors of locally variation, we apply it to a variety of datasets.

The first is natural images in grayscale taken by [68] and cropped to 16×16-pixel regions. We do not have labeled "ground-truth factors of variation" for this data, but we are able to provide qualitative results by inspecting the columns of the generator Jacobian matrix. This data was made publicly available without a specific license, so we analyze it under fair use.

The second is a dataset of simulated 64×64-pixel grayscale images of three black dots on a white background, inspired by [100]. The ground-truth factors of variation for this dataset are the x/y coordinates of the dot centers. We reimplement [100]'s code to generate the dot images and modify the code so that dots can overlap. We note for this dataset that if a model were to disentangle individual dot motions into different latent directions, then, by symmetry, we would expect identical singular values of J_g for those directions. Thus, we expect that for this dataset beta-VAEs will be unable to isolate individual dot motions since it has trouble disentangling directions in which the generator Jacobian has equal singular values.

Finally, we also apply our approach to tiled images of a real robotic arm taken from the MPI3D-real dataset [28], licensed under Creative Commons Attribution 4.0 International License. For each data point, we downsample four random images of the robot arm holding a large blue square in different locations and tile the random images in a 2×2 pattern to generate a new, more complicated 64×64-pixel image containing four different images of a real robotic arm. We call this tiled image dataset MPI3D-Multi.

4.5.2 Training

For the three-dots and MPI3D-Multi datasets, models are trained using a Bernoulli loss on batches of 64 images over a total of 300,000 batches. The

Adam optimizer is used with a learning rate of 0.0001 (matching [55]). We use linear annealing from 0 to the final hyperparameter value over the first 100,000 batches for both the beta hyperparameter and JL1-VAE's γ parameter in our implementations for JL1-VAE and beta-VAE (unlike [55]). Annealing is beneficial in avoiding model collapse when adding our L_1 regularization term. Models are trained on a Nvidia Quadro V100 hosted locally and one hosted on Google Cloud. Training each JL1-VAE model on ten latent variables takes approximately 2.5 hours, while training each beta-VAE model takes approximately 45 minutes. In total, training ten JL1-VAE models and ten beta-VAE models for quantitative evaluation takes roughly 33 hours.

Models are trained for 100,000 batches of 128 images for the natural image dataset. We use the Adam optimizer with a learning rate of 0.001 and train on a Nvidia Quadro V100s hosted locally. Training takes nine minutes for the beta-VAE and 23 minutes for the JL1-VAE.

4.5.3 Evaluation Metrics

Several metrics are commonly used to measure the "disentanglement" of a latent representation. In this chapter, we address two common metrics, the Mutual Information Gap (MIG) [15] and Modularity [78], and show how we are able to provide extensions to these metrics that give a measure of how well a representation *locally* disentangles factors of variation.

The original MIG and modularity metrics measure global disentanglement—that is, they measure across the whole dataset how well each latent variable maps to a unique ground-truth factor of variation. Since the JL1-VAE does not add an explicitly global disentanglement incentive to beta-VAEs, but instead is designed to locally encourage disentanglement using the Jacobian of the generative map, we do not expect it to improve the global disentanglement of factors of variation necessarily. For example, the JL1-VAE may assign factors of variation to latent variables using one pairing in one local region of the latent space and a different pairing in a different region of the latent space. This could lead to good average local disentanglement but would not lead to good global disentanglement.

We are therefore interested in defining *local* disentanglement metrics based on MIG and modularity. We call these metrics “local MIG” and “local modularity,” and the general form of their calculation is to compute each metric several times on different random “local” samples from the global dataset and then average the results. As with MIG and modularity, calculating these disentanglement metrics requires a generative model of the data from ground-truth factor values.

The key technique for each of our local metrics is to repeatedly compute the disentanglement metric on random local samples of data. To generate a random local sample of data, we randomly choose a centroid from the ground-truth factor values and then random sample N ground-truth datapoints within an L_∞ distance ρ from that centroid. The radius ρ is a hyperparameter determining how close ground-truth factors of variation need to be in order to be considered “local.” We scale the hyperparameter ρ as a fraction of each latent variable’s total range of available values. This set of N datapoints comprises each local data sample. For our experiments, we choose $N = 10,000$.

We apply the MIG and modularity metrics to each local sample of data to determine the disentanglement of the latent space in that local region. We use the MIG and modularity calculation implementations from the open-source (Apache License 2.0) `disentanglement_lib` library [55].

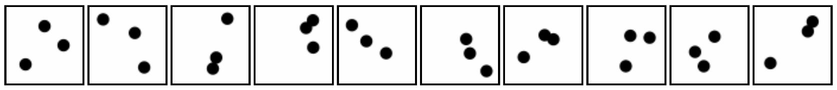
We repeat this algorithm with 20 different local data samples and report the average as the local disentanglement score.

4.6 JL1-VAE Results

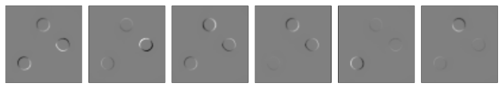
Qualitative results for the three-dots, MPI3D-Multi, and natural image datasets are shown, and quantitative results are presented for the three-dots dataset.

4.6.1 Qualitative Results

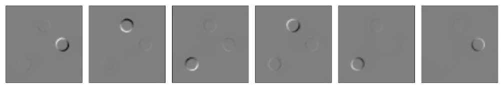
Qualitative results are generated by inspecting the generator Jacobian at the (deterministic) latent embeddings for example images. Each generative Jacobian matrix column is associated with a latent direction and shows how the generated image would change from a slight perturbation to the embedding in that latent direction. The generative Jacobian matrix columns for natural images are shown in Figure 4.1. There, we show the largest 20 PCA components, an arbitrary sample of 20 ICA components (from training 100 ICA components using FastICA [70]), and the ten Jacobian matrix columns for the beta-VAE and



(a) Example three-dots images



(b) Jacobian matrix columns for beta-VAE



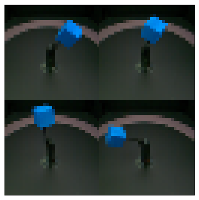
(c) Jacobian matrix columns for JL1-VAE

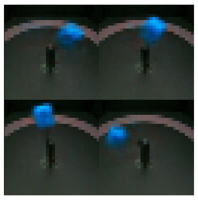
Figure 4.3: Qualitative results for the three-dots dataset. We show six Jacobian matrix columns for beta-VAE ($\beta = 4$) and JL1-VAE ($\beta = 4$, $\gamma = 0.1$) evaluated for the leftmost example image.

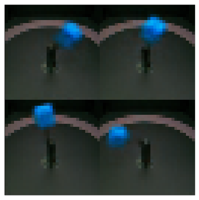
JL1-VAE models. Additional visualizations are included in the Appendix. We discern more localization (results more similar to local receptive fields) in the JL1-VAE and ICA results compared to the beta-VAE and PCA results.

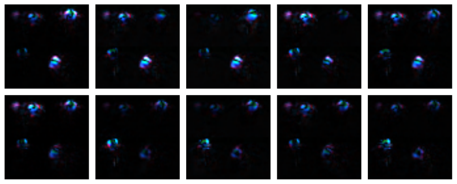
The same procedure was used for the three-dots dataset and MPI3D-Multi to generate qualitative results. Results for the three-dots dataset are shown in Figure 4.3. There, we show the six Jacobian columns with the largest L_2 norms for the three-dots dataset for both our JL1-VAE and beta-VAE (all ten Jacobian columns are visualized in the Appendix). Qualitatively, when evaluating the Jacobian of the generator function for our JL1-VAE, individual dot motions are separated into different latent components. The beta-VAE does not exhibit this behavior.

For the MPI3D-Multi dataset, containing tiled images of a real robot, the JL1-VAE again does a better job of separating the four robots in each image into separate latent variables. These results are presented in Figure 4.4.


(a) MPI3D-Multi image


(b) Beta-VAE reconstruction


(c) JL1-VAE reconstruction


(d) Jacobian matrix columns for a beta-VAE

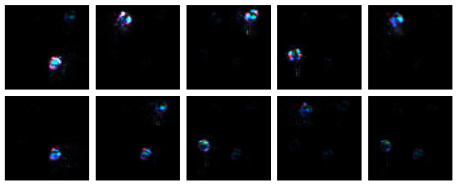

(e) Jacobian matrix columns for a JL1-VAE

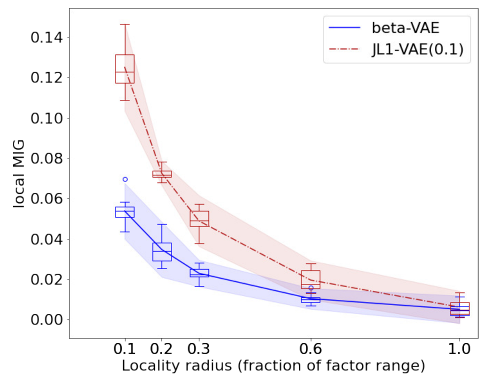
Figure 4.4: Qualitative results for MPI3D-Multi. JL1-VAE shows stronger pressure to locally disentangle individual robot motions. Both models used $\beta = 0.01$. For JL1-VAE, $\gamma = 0.01$.

4.6.2 Quantitative Results

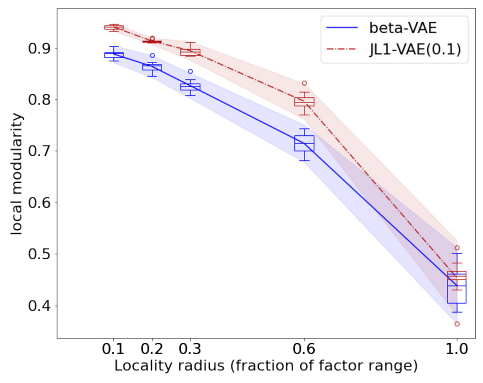
Local disentanglement scores are generated for the models trained on the three-dot images.

To observe the effect of the ρ parameter of our local disentanglement metrics, in Figure 4.5 we plot the varying local disentanglement scores as we change the ρ parameter for our JL1-VAE ($\beta = 4.0$ and $\gamma = 0.1$) and a standard beta-VAE ($\beta = 4.0$). The hyperparameter for β was chosen near the middle of the range used in [55]. We also tried other hyperparameter values and saw similar results. Too large a γ can lead to model collapse, so we chose a small enough γ to avoid that collapse but otherwise large enough to start to see reconstruction performance degradation, so we knew that its regularization was affecting model training.

We note that JL1-VAE attains higher local disentanglement scores compared to beta-VAEs, which is especially true as we look at more localized samples of data, corresponding to a smaller ρ parameter. For $\rho = 0.1$, we see significantly higher local disentanglement scores for JL1-VAE compared to beta-VAE ($p < 0.001$ for T-test), but for $\rho = 1$, testing for global disentanglement, we see indistinguishable disentanglement scores between the two ($p > 0.05$ for T-Test). Our JL1-VAE is able to *locally* disentangle factors of variation for the three-dots dataset, but does not globally disentangle factors of variation.



(a) Local MIG scores

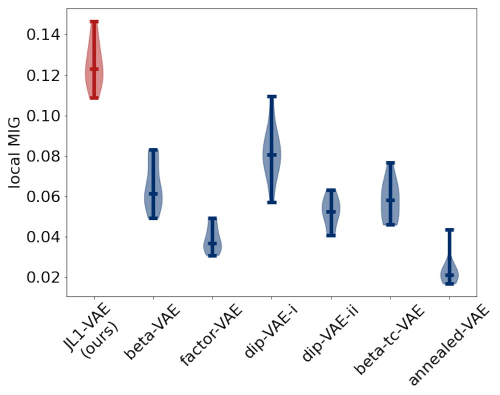


(b) Local modularity scores

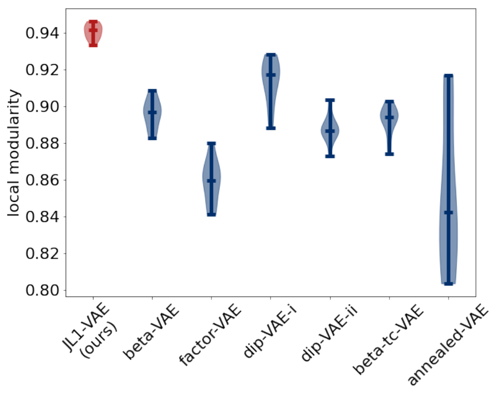
Figure 4.5: Local disentanglement scores varying the locality parameter ρ. Ten JL1-VAE and beta-VAE models were trained on the three-dots dataset with $\beta = 4$ and, for JL1-VAE, $\gamma = 0.1$.

Fixing the ρ parameter to 0.1, we also compute the local MIG and local modularity scores for six different comparative methods, namely beta-VAE, FactorVAE, DIP-VAE-I, DIP-VAE-II, β-TCVAE, and AnnealedVAE, using the implementations of `disentanglement_lib` with our convolutional architecture. A description of each of these models can be found in [55]. We trained ten iterations of those models with different random seeds using hyperparameters chosen near the middle of the suggested ranges in that work. That included training ten new beta-VAE models with new random seeds. All models were trained with ten latent dimensions.

Local MIG and local modularity scores are shown in Figure 4.6. We see a range of disentanglement scores due to the random seeds used to generate our models (ten models for each learning algorithm). Additionally, our local MIG and modularity metrics have some additional stochasticity due to the randomness in sampling 20 local samples of 10,000 points during the calculation of those metrics. Nevertheless, we observe significantly higher disentanglement scores ($p < 0.001$ for T-test) for our JL1-VAE compared to every baseline method.



(a) Local MIG scores



(b) Local modularity scores

Figure 4.6: Local disentanglement scores for JL1-VAE models and baseline implementations from [55]. The baseline implementations use default hyperparameters from that paper, choosing values near the middle when a range of hyperparameters are listed. Each model is run ten times with new random seeds. Local disentanglement is calculated using $\rho = 0.1$ with 20 different local samples.

4.7 Discussion of JL1-VAE

In this chapter, we presented JL1-VAE, a VAE augmented with an L_1 regularization to the Jacobian to improve local disentanglement. We extended the MIG and modularity disentanglement metrics to generate metrics measuring local disentanglement. We evaluated our model on natural images, simulated images of dots, and tiled images of a real robot, and we showed qualitatively and quantitatively that our method can improve local disentanglement in the generated representation. We believe that for trajectory data, the disentanglement will be according to different temporal regions of the trajectory, like the disentanglement here was over different pixel regions of the image.

Our added L_1 regularization of the Jacobian of the generator function is motivated by the use of L_1 regularization to prefer certain orientations in ICA and sparse coding, by a desire to relate each latent direction to sparser pixels (more similar to localized receptive fields), and by the implicit L_2 regularization already present in beta-VAEs. We show that this L_1 regularization term can encourage latent axes to align locally with ground-truth factors of variation. While this approach shows promise for local alignment, it does not address global alignment issues. That is, in one part of the dataset, the learned representation may assign a latent variable e_1 to follow a certain latent factor of variation. In a different part of the dataset, it might be a different latent variable e_2 that follows that latent factor of variation.

Regarding "no free lunch" theorems that show unsupervised disentanglement is impossible without inductive biases [55], we note that L_1 regularization of the generative Jacobian generates an inductive bias. The inductive bias encourages small axis-aligned perturbations of the latent space to result in sparse changes to the image space, whether that be expressed as localized receptive fields that act only on small regions of the image space, as seen in Figure 4.1, or motions of only a single dot or robot, as seen in Figures 4.3 and 4.4. Cases for which this inductive bias might not add value would include, for example, single robot images from MPI3D, where the primary ground-truth factors of local variation affect the same object within the same image patch or images where one factor of variation includes global lighting changes.

CHAPTER 5

CONCLUSION

5.1 Summary of Contributions

This work presented three approaches to improve how robots learn latent spaces for robotic applications. These improvements were better regularization of the latent space to avoid overfitting to the training data (CurvVAE), intentional separation of timing and spatial factors of variation when learning latent spaces of trajectory data (TrajectoryVAE), and local disentanglement of factors of variation that affect different regions of images (JL1-VAE). Together, these improvements are a step toward more useful and understandable latent variable models that can be applied in physical robotic domains. They are all improvements to low-dimensional latent variable models and provide ways for robots to model datasets in ways that can result in more useful (less overfit to the data) and more interpretable (with a simpler relationship between latent variables and the data) models. These contributions are all based on modifying the function from the latent space to the generated data, whether by adding a regularization term (CurvVAE, JL1-VAE, and TrajectoryVAE) or by also enforcing architectural constraints to force meaning on the latent values (TrajectoryVAE). Additionally, this dissertation presented several physical robot experiments to demonstrate the benefits of the presented contributions and give further evidence of the usefulness of latent variable models for robotic applications.

5.2 Potential Extensions of Presented Algorithms

In our motivational example of how a latent variable model could be used to select a trajectory for the robot to execute, we did not explore the possibility that the trajectory would not satisfy the desired task. Unplanned events often happen in robotics, so waiting until a full trajectory is complete before noticing task failure is generally inefficient. Using technical terminology, the fact that our proposed method and robot experiments rely on open-loop execution of the trajectories is an inefficiency in the current implementation of trajectory-based learning. While we could wait for the trajectory to finish before picking a new trajectory to execute and trying again, it would be more efficient for that process to happen during trajectory execution, perhaps even changing the trajectory rather than restarting the trajectory. Future work could incorporate feedback from the user or the environment during trajectory execution to detect and react to deviations from the desired outcome.

Our example use cases rely on a human picking from the low-dimensional latent space to choose a trajectory or an image. That use case gives a clear motivation for why learning a simple, elegant, interpretable latent variable model of arbitrary data is useful. However, presumably there are instances where these types of latent variable models would also make it easier for robots to choose trajectories to execute autonomously. Future work could explore whether improved latent variable models can help autonomous learning agents choose better trajectories.

We note that the JL1-VAE work was only applied to image data. We would like to see this approach applied to multimodal data gathered from a real robotic system to understand broader applications. For example, this data could potentially be applied to trajectory data to encourage disentanglement of different temporal regions of a trajectory (latent directions that might only affect the beginning or end of the trajectory) as opposed to different spatial areas of an image.

We note that many of these approaches will increase the training time of the latent space. For example, JL1-VAE currently calculates the full Jacobian of the generator during training, leading to training times that scale linearly with the number of latent dimensions. Future work could improve the regularization terms' efficiency through a more efficient sampling-based approach, whether by sampling only certain Jacobian columns during training or by somehow computing the regularization term less frequently.

Additionally, a practical limitation inherent in the proposed approaches to learning trajectories from human demonstration is the need to compute how the robot's actual joints should move in order for the robot's end-effector to match the desired demonstration motion. When speed is not a concern, this translation has a straightforward solution. Any inverse kinematics solver can solve the robot's initial pose. Then, the inverse Jacobian can be used to calculate the required joint changes to move the end-effector along the desired trajectory. However, when using a physical robot, it is easy to encounter situations where that approach will hit joint speed or torque limits. In cases with many solutions to the inverse kinematics problem, multiple solutions can be searched to find a feasible joint motion for the robot. For example, a 7-degree-of-freedom robot will generally have a free degree of freedom that can be used to adjust the position of the robot joints without changing the position or orientation of the end-effector. Likewise, suppose we only constrain the end-effector's position (not its orientation) on a four or higher degree-of-freedom robot arm. In that case, additional degrees of freedom can be changed without moving the end-effector position. The search over these degrees of freedom at each timestep in the trajectory in joint space to minimize the required torque to execute the desired end-effector motion in task space can be written as a large non-convex optimization problem with no elegant solution. Future work could improve how robots with extra degrees of freedom can learn to execute desired end-effector motions using minimal energy or time.

5.3 Broader Perspectives

There are currently many active areas of research in robotics and artificial intelligence. For manipulation tasks, one active area for teaching robots to perform complicated manipulation tasks is through learning from demonstration. Since many humans can easily perform the manipulation task we want the robot to learn, it makes sense to try to have the human transfer that knowledge to the robot by demonstrating how to perform the desired motions. The type of learning-from-demonstration approach presented in this dissertation has recently had a resurgence, suggesting that it is a good approach. For example, recent impressive advances in robotic manipulation, like Diffusion Policy [18] and RT-2 [102], rely on human demonstration data to teach the robot how to perform manipulation actions. This work focuses on ways to model human demonstration data cleanly using latent variable models. Many works have focused on robot learning without any human demonstration data. However, adding human demonstration data to those approaches is likely to accelerate robot learning even faster.

5.4 Greater Goal and Vision of this Work

The various components of this work come together toward the greater goal of having robots learn simple, accurate, and useful generative models. The structure of all these contributions has been to take some task, like generating movements to pick up food with a fork, and break it down into the simplest possible training dataset, like picking up a single banana slice from a particular known location, and then explore how we can build simple models of trajectories to solve that single task. We found interesting challenges on each task, like how to handle small dataset sizes (a challenge we tackled with CurvVAE), how to handle the timing of trajectories (a challenge we tackled with TimewarpVAE), and how to handle degeneracies in choice of rotation of the meaning of latent space directions (a challenge we tackled with JL1-VAE). Each of these approaches helped with particular challenges, but depending on the problem at hand, they can be combined as needed.

My vision of how these contributions could be best used in practice would be to construct a library of simple generative models to solve a variety of tasks, each trained on curated data to solve a particular task. For example, one model in the library could be the generative model to pick up banana slices off of a plate, like the model presented in the CurvVAE work. Another model in the library, like the model presented in the TimewarpVAE work, might be used to scrape food up along the contours of a bowl with a particular size and shape. One benefit of focusing the generated models on small tasks is that they can be validated through manual sampling of latent sweeps, as was done in this work, or potentially through more automated validation. The goal would be to check each individual generative model to confirm that it would conform to desired safety constraints (for example, staying within some desired workspace, below some maximal speed, etc.) as long as the input latent values were within some known safe bounds. The choice of model from the library could then be selected by the user (through some AI-assisted search through the library of validated models) and the choice of latent to execute could then also be selected by the user (again, through a potentially AI-assisted through the latent space). This vision of the user selecting and then executing a full motion is somewhat different from the type of manual control seen in shared autonomy [42] or a learned mapping from continual joystick inputs to robot actions [57] where the user is constantly providing control input. It is closer to how I imagine performing motions like striking a nail with a hammer or swinging a golf club, where almost all of the motion planning and selection happens before the execution. My view in this work is that the desired motion should be able to be explained to the robot and selected before execution and that further motion selections should not need to be made during the execution of the desired trajectory unless something unexpected happens.

5.5 Future Human User Studies

As part of this grand goal, it will be important and useful to continually test with human participants in future work. Questions that should be studied in the future include testing whether the learned latent spaces really are “interpretable” to human users. Can the user quickly select a desired trajectory by searching the latent space from the learned model? How quickly does a user become familiar with how the different latent directions affect the specified trajectory? Is it important that different models in the library maintain certain conventions for how latent values affect trajectories? How should the robot display the user’s specified trajectory back to the user? For the fork trajectory work, if the robot moves the fork to the starting pose of the trajectory, I am able to get a good feel for what the rest of the trajectory will look like. However, for more complicated trajectories, what type of display or projection best allows the user to understand the selected trajectory? What input modality works well for selecting the latent value? How much assistance in selecting latent value is helpful? These questions could be studied and evaluated using the framework presented in [37]. Similar questions have been addressed in [7] and would need to be reevaluated as enhancements are made to the robotic system. All these questions are critical for moving this work from research into practice and for developing this grand goal into a usable framework for real human use.

APPENDIX A
APPENDIX

A.1 Derivation of TimewarpVAE from Dynamic Time Warping

Dynamic time warping (DTW) compensates for timing differences between two trajectories by retiming the two trajectories so that they are spatially close to each other at matching timestamps. In this section, we explicitly derive TimewarpVAE from continuous dynamic time warping, the formulation of dynamic time warping for continuous functions presented by [52].

A.2 Continuous Dynamic Time Warping

We begin with a brief summary of continuous DTW. Given two trajectories x_0 and x_1 (each a function from time in $[0,1]$ to some position in $\mathbb{R}^n$), continuous DTW learns two time-warping functions, ρ_0 and ρ_1, where each time-warping function is monotonic and bijective from $[0,1]$ to $[0,1]$. The goal is to have the time-warped trajectories be near each other at corresponding (warped) timesteps. Mathematically, ρ_0 and ρ_1 are chosen to minimize the integral

$$\int_0^1 \left\| x_1\left(\rho_1(s)\right) - x_0\left(\rho_0(s)\right) \right\|^2 \frac{(\rho_0)'(s) + (\rho_1)'(s)}{2} \, ds \tag{A.1}$$

This integral is the distance between the trajectories at corresponding timesteps, integrated over a symmetric weighting factor.

This algorithm has a degeneracy, in that many different ρ_0 and ρ_1 will lead to equivalent alignments $\rho_1 \circ \rho_0^{-1}$ and will therefore have equal values of our cost function. This becomes relevant when generating new trajectories from the interpolated model, as it requires choosing a timing for the generated trajectory.

A.2.1 Reformulation of Continuous DTW

The optimization criterion of continuous DTW can be rewritten as follows: Given the time-warping functions ρ_0 and ρ_1 from above, define $\phi_0 = \rho_0^{-1}$ and $\phi_1 = \rho_1^{-1}$. Let the function $f : [0,1] \times [0,1] \to \mathbb{R}^n$ be defined as $f(s,z) = (1-z)x_0(\rho_0(s)) + zx_1(\rho_1(t))$. That is, $f(s,z)$ is the unique function that is linear in its second parameter and which satisfies the boundary conditions $f(\phi_0(t),0) = x_0(t)$ and $f(\phi_1(t),1) = x_1(t)$. These boundary conditions associate x_0 with $z_0 = 0$ and x_1 with $z_1 = 1$.

Our minimization objective for choosing ρ_0 and ρ_1 (or, equivalently, for choosing their inverses ϕ_0 and ϕ_1) can be written in terms of f as

$$\frac{1}{2}\int_0^1 \left\| \left.\frac{\partial f(s,z)}{\partial z}\right|_{s=\phi_0(t),z=0} \right\|^2 dt + \frac{1}{2}\int_0^1 \left\| \left.\frac{\partial f(s,z)}{\partial z}\right|_{s=\phi_1(t),z=1} \right\|^2 dt. \tag{A.2}$$

The derivation goes as follows: We define $\phi_0 = \rho^{-1}$ and $\phi_1 = \rho^{-1}$, and we define the function $f : [0,1] \times [0,1] \to \mathbb{R}^n$ to be the unique function that is linear in its second parameter and which satisfies the boundary conditions $f(\phi_0(t),0) = x_0(t)$ and $f(\phi_1(t),1) = x_1(t)$. Equivalently, it satisfies the boundary conditions $f(s,0) = x_0(\phi_0^{-1}(s))$ and $f(s,1) = x_1(\phi_1^{-1}(s))$.

Substituting these definitions gives an optimization criterion of

$$\int_0^1 \left\|x_1\left(\phi_1^{-1}(s)\right) - x_0\left(\phi_0^{-1}(s)\right)\right\|^2 \frac{\left(\phi_0^{-1}\right)'(s) + \left(\phi_1^{-1}\right)'(s)}{2}\, ds \tag{A.3}$$

The boundary conditions of f imply this is equal to

$$\frac{1}{2}\int_0^1 \|f(s,1) - f(s,0)\|\, d\phi_0^{-1}(s) + \frac{1}{2}\int_0^1 \|f(s,1) - f(s,0)\|\, d\phi_1^{-1}(s) \tag{A.4}$$

And now, performing a change-of-variables $u = \phi_0^{-1}(s)$ and $v = \phi_1^{-1}(s)$ gives

$$\frac{1}{2}\int_0^1 \|f(\phi_0(u),1) - f(\phi_0(u),0)\|^2\, du + \frac{1}{2}\int_0^1 \|f(\phi_1(v),1) - f(\phi_1(v),0)\|^2\, dv \tag{A.5}$$

Since f is linear in its second coordinate, we can write this in terms of the partial derivatives of f

$$\frac{1}{2}\int_0^1 \left\| \left.\frac{\partial f(s,z)}{\partial z}\right|_{s=\phi_0(u),z=0} \right\|^2 du + \frac{1}{2}\int_0^1 \left\| \left.\frac{\partial f(s,z)}{\partial z}\right|_{s=\phi_1(v),z=1} \right\|^2 dv \tag{A.6}$$

A.2.2 Simultaneous Time-Warping and Manifold Learning on Trajectories

The relation to TimewarpVAE is as follows:

For each trajectory x_i, TimewarpVAE learns a low-dimensional latent representation $z_i \in \mathbb{R}^\ell$ associated with that trajectory. These z_i are the natural generalizations of the reformulation of continuous DTW above, which had hard-coded latent values $z_0 = 0$ and $z_1 = 1$ for the two trajectories.

For each trajectory x_i, TimewarpVAE learns a time-warping function ϕ_i that transforms timesteps to new canonical timings. These ϕ_i are the natural extension of the ϕ_0 and ϕ_1 from above.

TimewarpVAE learns a generative function f which, given a canonical timestamp s and a latent value z, returns the positon corresponding to the trajectory at that time. This is an extension of the function f, with relaxations on the linearity constraint and the boundary conditions. Instead of requiring f to be linear in the z argument, we parameterize f with a neural network and regularize it to encourage f to have small partial derivative with respect to the latent variable z. This regularization is described in Section A.2.3. Instead of a boundary constraints requiring $f(\phi_i(t), z_i)$ to be equal to $x_i(t)$, we instead add an optimization objective that $f(\phi_i(t), z_i)$ be close to $x_i(t)$,

A.2.3 TimewarpVAE Regularization of the Decoder

Training the decoder using our optimization objective includes adding noise ϵ to the latent values z before decoding. This encourages the decoder to take on similar values for nearby values of z. In particular, as described by [54], this will add an implicit Jacobian squared regularization of the decoder over the z directions. Penalizing these $\|\frac{\partial f(s,z)}{\partial z}\|^2$ terms is exactly what we want for our manifold-learning algorithm. Additionally, we note that we do not add any noise to the temporal encoder when computing the reconstruction loss, so our beta-VAE style architecture does not include any unwanted regularization of $\|\frac{\partial f(s,z)}{\partial s}\|^2$.

A.3 Degeneracy of Time Warping Using Notation of [98]

Using the notation of [98], the timing degeneracy noted in the main paper also applies. For any set of time-warping functions T^{θ_i}, one for each of the N_k different trajectories u_i with class label y_i out of K possible class labels, all of those time-warping functions can be composed with some additional fixed diffeomorphic warping T^p without changing the Inverse Consistency Averaging Error.

That is, composing all time-warps with a time-warping function T^p to generate $\tilde{T}^{\theta_i} = T^{\theta_i} \circ T^p$ will not affect the Inverse Consistency Averaging Error, since now the (perturbed) average warped trajectory for cluster k $\tilde{\mu}_k$ is the warp of the previous average warped trajectory μ_k

$$\tilde{\mu}_k = \frac{1}{N_k} \sum u_i \circ T^{\theta_i} \circ T^p = \frac{1}{N_k} \left(\sum u_i \circ T^{\theta_i} \right) \circ T^p = \mu \circ T^p \tag{A.7}$$

.

The inverse $\tilde{T}^{-\theta_i}$ is simply $T^{-p} \circ T^{-\theta_i}$

The Inverse Consistency Averaging Error using those perturbed time-warps $\tilde{T}$ is the same as that computed using the original time-warps.

$$\mathcal{L}_{ICAE}(\tilde{T}) = \sum_{k=1}^{K} \frac{1}{N_K} \sum_{i:y_i=k} \left\| \tilde{\mu}_k \circ \tilde{T}^{\theta_i} - u_i \right\|_{\ell_2}^2 \tag{A.8}$$

$$= \sum_{k=1}^{K} \frac{1}{N_K} \sum_{i:y_i=k} \left\| \mu_k \circ T^p \circ T^{-p} \circ T^{-\theta_i} - u_i \right\|_{\ell_2}^2 \tag{A.9}$$

$$= \sum_{k=1}^{K} \frac{1}{N_K} \sum_{i:y_i=k} \left\| \mu_k \circ T^{-\theta_i} - u_i \right\|_{\ell_2}^2 \tag{A.10}$$

$$= \mathcal{L}_{ICAE}(T) \tag{A.11}$$

This shows that there is a degeneracy over the choice of warping of the warped trajectories. Warped trajectories can all be additionally warped by another time-warping function without changing the Inverse Consistency Averaging Error.

A.4 TimewarpVAE Time-Warper Effect on Trained Model

In Fig. A.1 we show how varying the time-warper parameters affects the timing of the generated trajectory for one of the TimewarpVAE letter models. Here, we choose five different sets of time-warper parameters, and plot the resulting time-warper function $\phi(t)$. We then decode the same spatial trajectory using those five latent parameters. The resulting trajectories spatially would look all the same, and we plot the X and Y locations of the generated trajectories as a function of time. The associated timings of the trajectories changes due to the time-warper function.

A.5 TimewarpVAE Additional Interpolations for Models

Here, we present additional interpolation results, all on 16 latent dimensions and $\beta = 0.001$. We note that the convolutional encoder/decoder architecture in beta-VAE in Fig. A.2b does not appear to have as strong an implicit bias toward smooth trajectories as the TimewarpVAE architecture. This makes sense because the TimewarpVAE architecture decomposes the generative function into a component $g(s)$ which computes poses as a function of time, likely inducing

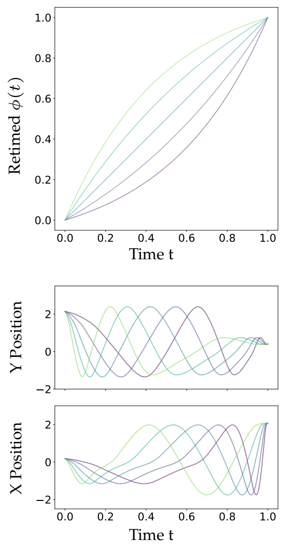


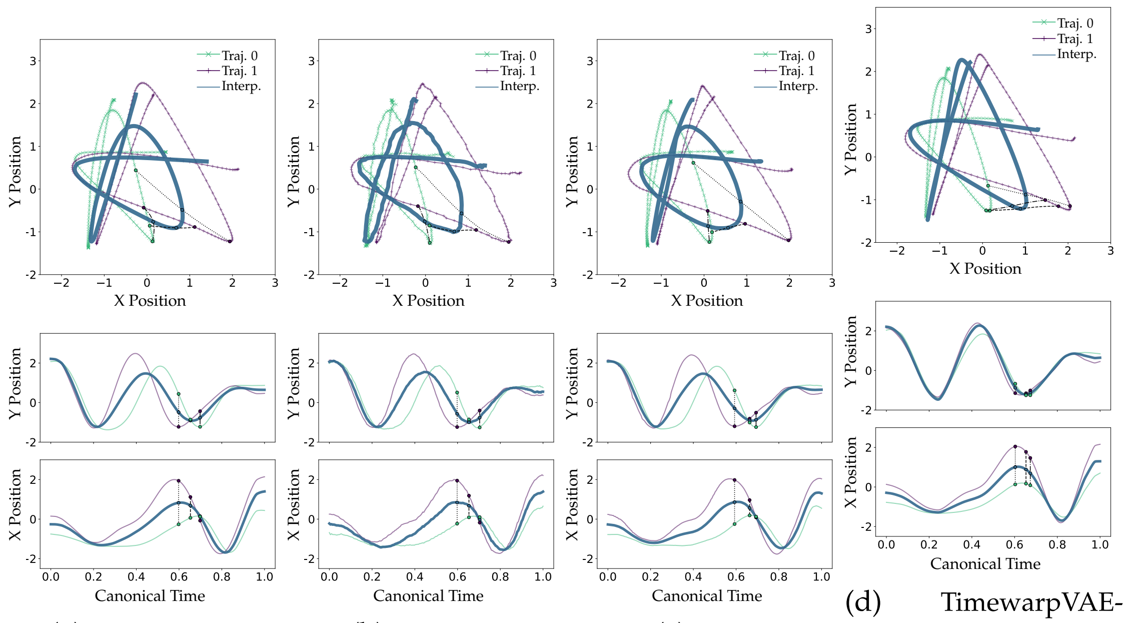
Figure A.1: Decoding the same spatial latent variable using different time-warping parameters will give the same spatial trajectory but with different timings (fast or slow at different times). We plot the time-warping functions for five different timing latents and the generated trajectories for a single spatial latent value by showing the generated positions as a function of time.

an inductive bias toward smoother trajectories as a function of time.

The interpolation in Fig. A.2a shows that in the ablation of TimewarpVAE without the timing module the interpolation does not preserve the style of the "A". Likewise, the DMP interpolation in Fig. A.2c does not preserve the style of the "A".

A.6 TimewarpVAE Equivalence of Regularizing ϕ or ϕ^{-1}

We note that our regularization is the same, regardless if we regularize ϕ or ϕ^{-1}. We show this by applying the substitution $s = \phi(t)$, $ds = \phi'(t)dt$. That substitution gives: $\int_0^1 \left(1 - \frac{1}{\phi'(\phi^{-1}(s))}\right) \log\left(\phi'(\phi^{-1}(s))\right) ds$.



(a) NoTimewarp (b) beta-VAE (c) DMP (d) TimewarpVAE-DTW

Figure A.2: Additional interpolation results

We use the identity $(\phi^{-1})'(s) = \frac{1}{\phi'(\phi^{-1}(s))}$ to simplify to

$$\int_0^1 \left(1 - (\phi^{-1})'(s)\right) \log\left(\frac{1}{(\phi^{-1})'(s)}\right) ds = \int_0^1 \left((\phi^{-1})'(s) - 1\right) \log\left((\phi^{-1})'(s)\right) ds \tag{A.12}$$

This is exactly our regularization applied to the function ϕ^{-1}. Thus, we note that our regularization is symmetric. Our regularization cost is the same whether it is applied to ϕ (the function from trajectory time to canonical time) or applied to ϕ^{-1} (the function from canonical time to trajectory time).

It didn't have to be that way. For example, if our cost function were of the form $\int_0^1 \log^2(\phi'(t))dt$, the substitution above would give a cost $\int_0^1 \frac{1}{\phi'(\phi^{-1}(s))} \log^2(\phi'(\phi^{-1}(s)))ds$ which simplifies to $\int_0^1 \frac{1}{\phi'(\phi^{-1}(s))} \log^2((\phi^{-1})'(s))ds$

which equals $\int_0^1 (\phi^{-1})'(s) \log^2((\phi^{-1})'(s))ds$ which is different from applying the regularization procedure to ϕ^{-1} which would have given a regularization term of: $\int_0^1 \log^2((\phi^{-1})'(s))ds$

A.7 Inspiration for Regularization Function

Our regularization function was inspired by Unbalanced Optimal Control. If we assign a uniform measure $\mathcal{U}$ to $[0, 1]$, we note that our regularization cost is exactly the symmetric KL Divergenge between $\mathcal{U}$ and the pushforward $\phi_{i*}\mathcal{U}$. For each ϕ_i, the pushforward $\phi_{i*}\mathcal{U}$ has a probability density function $1/(\phi_i'(\phi_i^{-1}(s)))$, and the symmetric KL divergence cost $D_{\mathrm{KL}}(\mathcal{U}|\phi_{i*}\mathcal{U}) + D_{\mathrm{KL}}(\phi_{i*}\mathcal{U}|\mathcal{U})$ gives the loss given above.

We work through the explicit mathematics below.

A.7.1 Pushforward of Probability Density Function

If F_0 is some cumulative distribution function, and F_1 is the CDF generated by the pushforward of a function ϕ then we have the simple identity $F_1(\phi(t)) = F_0(t)$. Taking the derivative of both sides with respect to t gives $F_1'(\phi(t))\phi'(t) = F_0'(t)$. The substitutions $s = \phi(t)$ and $\phi^{-1}(s) = t$ give $F_1'(s) = \frac{1}{\phi'(\phi^{-1}(s))} F_0'(\phi^{-1}(s))$. Since the PDF is the derivative of the CDF, then writing f_0 and f_1 as the corresponding PDFs of F_0 and F_1, we see

$$f_1(s) = \frac{1}{\phi'(\phi^{-1}(s))} f_0(\phi^{-1}(s)) \tag{A.13}$$

In the simple case where $f_0 = \mathcal{U}$, the uniform distribution over $[0,1]$, then $f_0(t) = 1$, so we have our pushforward

$$(\phi_* \mathcal{U})(s) = \frac{1}{\phi'(\phi^{-1}(s))} \tag{A.14}$$

A.7.2 Symmetric KL Divergence Calculation

The KL divergence $D_{\mathrm{KL}}(\mathcal{U}|\phi_*\mathcal{U})$ is $\int_0^1 \log\left(\phi'(\phi^{-1}(s))\right) ds$. Substituting $\phi(t) = s$ and the corresponding $\phi'(t)dt = ds$ gives $\int_0^1 \phi'(t) \log\left(\phi'(t)\right) dt$. Likewise, the KL divergence $D_{\mathrm{KL}}(\phi_*\mathcal{U}|\mathcal{U})$ is $\int_0^1 \frac{1}{\phi'(\phi^{-1}(s))} \log\left(\frac{1}{\phi'(\phi^{-1}(s))}\right) ds$, which simplifies with the same substitutions to $-\int_0^1 \log\left(\phi'(t)\right) dt$.

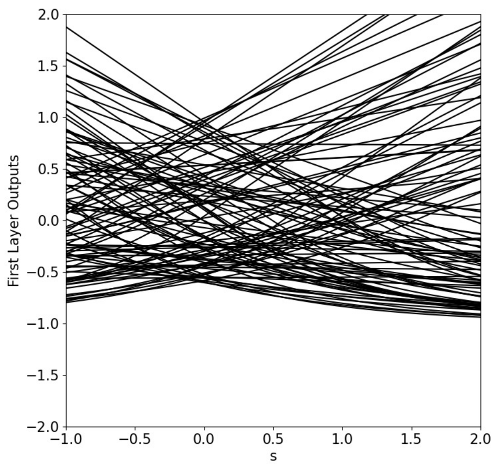
The symmetric KL divergence cost is thus

$$D_{\mathrm{KL}}(\mathcal{U}|\phi_*\mathcal{U}) + D_{\mathrm{KL}}(\phi_*\mathcal{U}|\mathcal{U}) = \int_0^1 \left(\phi'(t) - 1\right) \log\left(\phi'(t)\right) dt \tag{A.15}$$

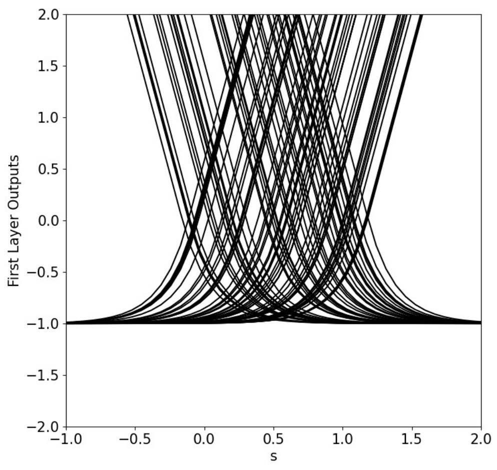
A.8 Initialization of Neural Network for $f(s, z)$

As mentioned above, the neural network $f(s, z)$ is split into $\mathbf{T}(z)$ and $g(s)$. Since $g(s)$ takes in a canonical time $s \in [0, 1]$ which we want to have roughly uniform modeling capacity over the full range of s from 0 and 1, we initialize the first layer of the neural network's weights, W (a matrix with one column), and bias b (a vector) for $g(s)$ in the following way.

We initialize the values in W to be randomly, independently $-G$ or G with equal probability, where G is a hyperparameter.



(a) Default PyTorch Initialization



(b) Custom Initialization

Figure A.3: Outputs of first layer of the neural network $g(s)$.

We then choose values of b so that, for each (output) row j, which we denote W_j and b_j the y-intercepts of the function $y = g_j(s) = W_j s + b_j$ are each an independent uniformly random value in $[0 - \eta, 1 + \eta]$

Visually, the effect of this initialization can be seen by plotting the first layer's transformation $\mathrm{ELU}(Ws+b)$ using PyTorch's default initialization and using our proposed initialization, where ELU is the Exponential Linear Unit introduced by [20]. PyTorch's default implementation randomly initializes the output functions to be distributed symmetrically around $s = 0$. Additionally, much of the modeling capacity is assigned to variations outside the domain $[0, 1]$. Since we know that the input timestamp will be $s \in [0, 1]$, our initialization focuses the modeling capacity near $[0, 1]$ and is symmetric around the middle of that range $s = 0.5$.

A.9 TimewarpVAE Timing Noise Data Augmentation

For data augmentation of timing noise, we create perturbed timesteps to sample the training trajectories as follows. First, we construct two random vectors ν_{in} and ν_{out} of uniform random numbers between 0 and 1, each vector of length 10. We then square each of the elements in those vectors and multiply by a noise hyperparameter η and take the cumulative sum to give perturbation vectors for input and output timings. We add each of those vectors to a vector with ten elements with linear spacing, $[0, 1/9, 2/9, 3/9, \ldots, 1]$. We then normalize those perturbed vectors so that the last elements are again 1. The resulting vectors now give nicely symmetric, monotonic x and y coordinates of knots for a stepwise-linear perturbation vector which we can subsample at arbitrary timesteps to give desired noise-added output timesteps. We choose $\eta = 0.1$ in our experiments, giving noise functions plotted in Fig. A.4.

When using this data augmentation, each time we pass a training trajectory into our model, instead of sampling the training trajectory at T uniformly-distributed timesteps between 0 and 1 to construct our $x \in \mathbb{R}^{T \times n}$, we first perturb all those timesteps by passing them through a randomly generated noise functions. This means that each x we pass into our learning algorithm has a slightly different timing than the training data, allowing us to perform data augmentation on the timings of the training data.

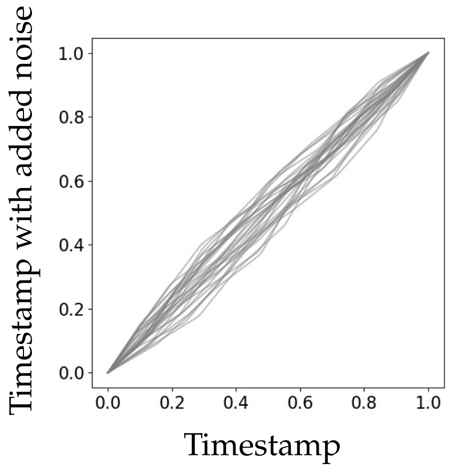
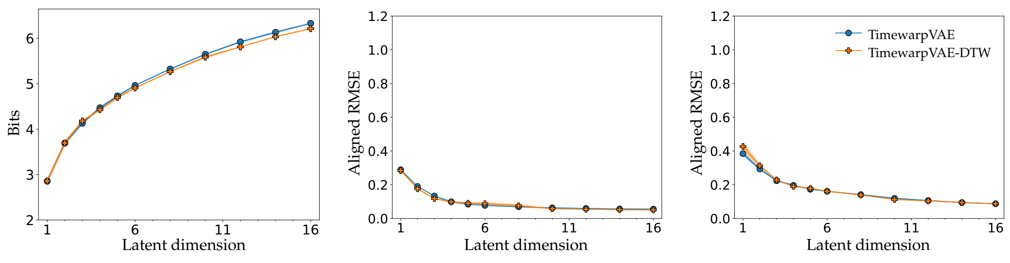


Figure A.4: Example (random) functions used to add timing noise during data augmentation

A.10 Alternative TimewarpVAE Approach Directly Using Dynamic Time Warping

An alternative formulation of TimewarpVAE, which we call TimewarpVAE-DTW, is to collect the decoded trajectory as a vector by running the decoder $f(s, z)$ over multiple, evenly-sampled canonical timesteps s, and then warping those generated timesteps to the training data using DTW. This is more similar to Rate Invariant AE, in that the time-warping happens after trajectory generation, however, we do not require linear interpolation. Instead, we convert the DTW alignment into a loss in such a way that all canonical trajectory points are used and averaged (using the same weightings we descriped for our Aligned RMSE). This avoids the problem encountered in Rate Invariant AE where parts of the canonical trajectory can be completely ignored. TimewarpVAE-DTW requires running DTW to align each reconstructed trajectory to its associated



(a) Rate (b) Train Distortion (c) Test Distortion

Figure A.5: Performance comparison between TimewarpVAE and TimewarpVAE-DTW

training trajectory every time the decoder function is executed during training. This is significantly less efficient than our suggested implementation of TimewarpVAE, because it requires many executions of dynamic time warping with no re-use of the DTW results between training steps. Our suggested implementation, TimewarpVAE, explicitly models the time-warping, so is able to re-use (and update) the warping function between steps, rather than recalculating it from scratch each time. However, we note in Fig.A.5 that TimewarpVAE-DTW, though less efficient, can give comparable results. In this implementation we use DTW, but a Soft-DTW [21] could be used instead.

A.10.1 TimewarpVAE Hyperparameters

The specific training architectures we use is shown in Table A.1. We use a kernel size of 3 for all convolutions. e is the spatial encoding architecture (which is always reshaped to a vector and followed by two separate fully connected layers, one outputting the expected encoding, and one outputting the log of the diagonal of the covariance noise). h is the temporal encoding architecture, which is always followed by a fully connected layer outputting a vector of size $K = 50$. $g(s)$ is part of the factorized decoder architecture, which is followed by a fully connected layer outputing a vector of size $m = 64$. $\mathbf{T}(z)$ is the other part of the factorized decoder architecture, which is followed by a fully connected layer outputting a vector of size nm where n is the number of channels in the training data (2 for handwriting, 7 for fork trajectories) and $m = 64$. For beta-VAE, instead of the factorized decoder architecture, we use one fully connected layer with output size 800, which we then reshape to size 25×32. This is followed by one-dimensional convolutions. Following the approach of [53], for the convolutions in the beta-VAE architecture, instead of doing convolutional transposes with strides to upsample the data, we instead always use a stride length of 1 and upsample the data by duplicating each element before performing each convolution. Thus, the lengths expand from 25 in the input to the output size of 200 after the three convolutions.

We use a learning rate of 0.0001, a batch size of 64, and a rectified linear unit (ReLU) for all spatial and temporal encoder nonlinearities, except for Rate Invariant AE for which follow the literature and we use Tanh. We use an exponential linear unit (ELU) for the decoder nonlinearities for TimewarpVAE, we use ReLU for the decoder nonlinearities for beta-VAE, and we again use Tanh for the Rate Invariant AE. We choose a variance estimate of $\sigma_R^2 = 0.01$ for our data, but this hyperparameter is not critical, as it is equivalent to scaling β and λ in our TimewarpVAE objective. In order to compute the Rate (information bottleneck) of the Rate Invariant AE, we implement it as a VAE instead of an autoencoder, only adding noise to the spatial latent, not to the timing latent values. We use 199 latent variables for the timing (one fewer than the trajectory

Table A.1: Hyperparameters

Name	e conv channels strides	h conv channels strides	$g(s)$ fc	$\mathbf{T}(z)$ fc	f fc conv channels
TimewarpVAE	[16,32,64,32] [1,2,2,2]	[16,32,32,64,64,64] [1,2,1,2,1,2]	[500,500]	[200]	–
NoTimewarp	[16,32,64,32] [1,2,2,2]	–	[500,500]	[200]	–
NoNonlinearity	[16,32,64,32] [1,2,2,2]	[16,32,32,64,64,64] [1,2,1,2,1,2]	[500,500]	[]	–
beta-VAE	[16,32,64,32] [1,2,2,2]	–	–	–	[800] [20,20,n]
Rate Invariant AE	[32,32,32] [1,1,1]	–	–	–	[6400] [32,32,n]

length), and vary the number of spatial latent variables.

A.11 TimewarpVAE Robot Execution

Here we give specifics on the execution of the TimewarpVAE trajectory on the Kinova Gen3 robot arm.

A.11.1 Optimization Formulation

We formulate an optimization problem to find the fastest feasible trajectory that satisfies the joint speed an torque constraints of our Kinova Gen3 robot arm and follows the training demonstration trajectory within a defined time-warping cost. We use Drake [89] to efficiently formulate several of the constraints, including the forward dynamics of the manipulator arm.

We write the time-warping function ϕ now as a function from real time $[0, T]$

to canonical time $[0, 1]$. Given the target canonical path $p(s) \ : \ [0, 1] \to \mathbb{R}^3$, and our robot joints as a function of the real time $j(t) \ : \ [0, T] \to \mathbb{R}^7$ and our forward robot kinematics mapping joint angles to end-effector position $f(j) \ : \ \mathbb{R}^7 \to \mathbb{R}^3$. Our first constraint is that the robot must follow the canonical path according to the time-warping funciton

$$p(\phi(t)) = f(j(t)) \tag{A.16}$$

We also constrain the time-warping function ϕ to be monotonically positive from $[0, T]$ to $[0, 1]$ by constraining its slope to be positive and for ϕ to have the right boundary constraints:

$$\phi'(t) > 0 \tag{A.17}$$

$$\phi(0) = 0 \tag{A.18}$$

$$\phi(T) = 1 \tag{A.19}$$

We define the forward kinematics funciton f by loading the Kinova Gen3 URDF into Drake. Likewise, we model the forward dynamics using drake, so that, under input joint torques at time $t$given by $\tau(t) \ : \ [0, T] \to \mathbb{R}^7$ we know the joint accelerations $j''(t)$ as a funciton of time. The forward dynamics gives us a differential equation of $j''(t)$ as a funciton of j', j, and τ. Using FD as the forward dynamics function, we have the constraint

$$j''(t) = FD(j'(t), j(t), \tau(t)) \tag{A.20}$$

We additionally add joint speed limits $j'_{i\max}$ for each joint i.

$$-j'_{i\max} \leq j'_i(t) \leq j'_{i\max} \tag{A.21}$$

We also add torque constraints for each joint i

$$-\tau_{i\max} \leq \tau_i(t) \leq \tau_{i\max} \tag{A.22}$$

Finally, we add a time-warping constraint, that our time-warping regularization value must be less than some bound. We note that we adjust the definition of our time-warping cost to account for the new domain of the time-warping of $[0, T]$ instead of $[0, 1]$. Our time-warping regularization value now contains additional factors of T and $\frac{1}{T}$ so that it is not affected by linear scaling of the real time duration T.

$$\frac{1}{T}\int_0^T (T\phi'(t) - 1)\log(T\phi'(t))\,dt \tag{A.23}$$

and we constrain this to be below some value C_{warp}

$$\frac{1}{T}\int_0^T (T\phi'(t) - 1)\log(T\phi'(t))\,dt \leq C_{\text{warp}} \tag{A.24}$$

Our optimization objective is to minimize T (how long it takes the robot to perform the motion) subject to all the constraints above.

A.11.2 Optimization Using Direct Collocation

To numerically solve the optimization problem, we use the direct collocation appraoch of [34] implemented in Drake. Rather than requiring that all the constraints above be satisfies at all timesteps, we only check constraints at fixed, equally-spaced timesteps. In particular, we approximate $j(t)$,$j'(t)$,and $\phi(t)$ as cubic splines (with K segments) and $K + 1$ equally spaced knots (including the knots at the endpoints at 0 and T), and we constrain the derivative of the cubic spline $j(t)$ to equal the cubic spline $j'(t)$ at the knot and at the collocation points (which are the midpoints in between two knots). We restrict $\tau(t)$ and $\phi'(t)$ to be piecewise affine with the same equally-spaced knots, and we now just check the following constraints at knot points and collocation points:

$$p(\phi(t)) = f(j(t)) \tag{A.25}$$

$$-j'_{i\text{max}} \leq j'_i(t) \leq j'_{i\text{max}} \tag{A.26}$$

$$-\tau_{i\text{max}} \leq \tau_i(t) \leq \tau_{i\text{max}} \tag{A.27}$$

$$j''(t) = FD(j'(t), j(t), \tau(t)) \tag{A.28}$$

The following constraints are only checked at knot points:

$$\phi'(t) > 0 \tag{A.29}$$

And, of course, the following constraints are still just at the boundary points:

$$\phi(0) = 0 \tag{A.30}$$

$$\phi(T) = 1 \tag{A.31}$$

The warping cost constraint is now approximated by a trapizoidal integration over the $K + 1$ knot points $t_k \in \{0, \ldots, T\}$:

$$\sum_{t_k} w_k \left(T\phi'(t_k) - 1\right) \log(T\phi'(t_k)) \leq C_{\text{warp}} \tag{A.32}$$

where the weighting w_k is $\frac{1}{2K}$ if t_k is 0 or T (a boundary) and $\frac{1}{K}$ otherwise.

We then optimize using IPOPT [95], terminatng after 2000 max iterations. We use $K = 20$ cubic spline segments. For NoWarping, we set the max time-warping cost C_{warp} to zero, and when we allow warping, we set the max time-warping allowed to 0.05. Based on the Kortex Gen3 User Guide, we set the joint speed limits to 1.39 rad/s for the first four joints and 1.22 rad/s for the last three joints. Likewise, we set the joint torque limits to 32 Nm for the first four joints and 13 Nm for the last three joints.

A.11.3 TimewarpVAE Robot Control

For robot execution, we send low-level joint commands at 500Hz to the Kinova Gen3 arm, using a PID controller to follow the desired joint position and velocities. The low-level commands directly specify the current desired for each joint, since we found there to be an unusably high delay when commanding via the TORQUE_HIGH_VELOCITY low-level API provided by Kinova, leading to controller instability. The TORQUE low-level API is implemented by Kinova leads to stable control, but contains a cascading controller that includes a velocity control loop, and therefore does not match the expected relationship between commanded torque and joint accellerations (in particular, commanding a robot joint to move with maximal torque accelerates the joint much more slowly than it would without Kinova's cascading controller). For these reasons, we use the MOTOR_CURRENT low-level API in a PID loop to command the Kinova Gen3 arm to track the joint trajectory returned from our optimization calculation.

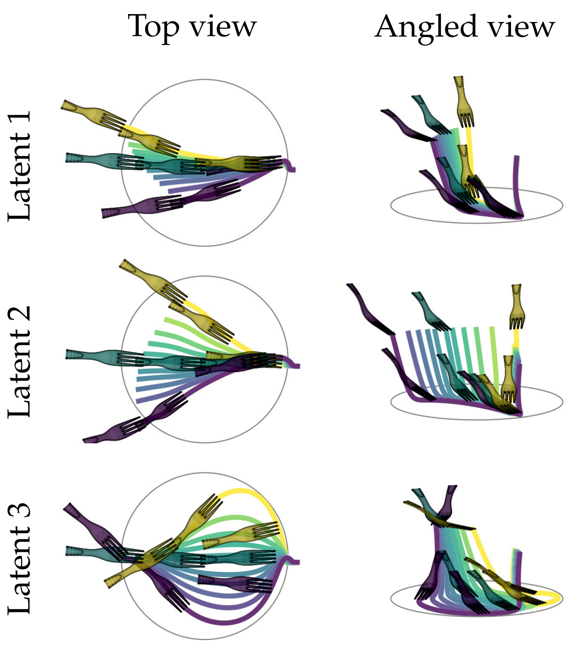


Figure A.6: Latent sweeps of trajectories. Latents 1 and 2 primarily affect the starting position. Latent 3 primarily affects how the trajectory bends.

A.11.4 Fork Additional Experimental Model's Learned Latent Dimensions

The model trained for the yarn acquisition experiments happened to have a slightly different architecture and hyperparameters. However, we can still associate meaning to the different latent directions, showing robustness to our approach. Here, the first two latent directions primarily affect the two-dimensional coordinates of the starting position of the fork in the X-Y plane. The third latent direction primarily affects the curvature of the trajectory, whether it angles to the left or the right before scraping the yarn to the pickup location on the plate. A sweep through these three different values is shown from above and from an angle in Fig. A.6. The choice of which latent value affects which component of the trajectory is arbitrarily learned by the model (there is perturbation symmetry in the training setup). This learned model can be viewed as a rotation of the latent space of the previously learned model, where the first two latents affect the starting position in different directions than before.

A.12 JL1-VAE Neural Network Architecture

We use a convolutional neural network architecture for our models. Our code can be found in our open source repository.[1]

For the autoencoders used on 64$\times$64-pixel images, we mimic the architecture presented in [55]. A 4$\times$4 kernel was used for all convolutional layers, with a stride of 2. A ReLU was used between all layers, with a final sigmoidal layer on the reconstruction architecture and a Bernoulli loss. In Tables A.2 and A.3, "Conv2d" refers to a convolutional layer, "FC" refers to a fully connected layer, "ConvT2d" refers to convolutional transpose, and the "($\times$ 2)" in the embedding architecture refers to the separate mean and log variance heads on the shared architecture.

For the autoencoders used on 16$\times$16-pixel images (the natural image crops), 3$\times$3 kernels were used for all convolutional layers and a stride of 2 everywhere except for the last reconstruction layer, which has a stride of 1. ReLU was used between all layers, with a final sigmoidal layer on the reconstruction architecture and a Bernoulli loss.

[1]https://github.com/travers-rhodes/jlonevae

Table A.2: Embedding and reconstruction architectures for 64×64-pixel images

Embedding	**Reconstruction**
Input: 64×64, 1 or 3 channels	Input: 10 values
Conv2d: 32 channels	FC: 256 channels
Conv2d: 32 channels	FC: 4×4 image, 64 channels
Conv2d: 64 channels	ConvT2d: 64 channels
Conv2d: 64 channels	ConvT2d: 32 channels
FC: 256 channels	ConvT2d: 32 channels
FC (× 2): 10 values	ConvT2d: 64×64, 1 or 3 channels

Table A.3: Embedding and reconstruction architectures for 16×16-pixel images

Embedding	**Reconstruction**
Input: 16×16, 1 channel	Input: 10 values
Conv2d: 64 channels	FC: 128 channels
Conv2d: 128 channels	FC: 4×4 image, 64 channels
FC: 128 channels	ConvT2d: 64 channels
FC (× 2): 10 values	ConvT2d: 32 channels
	ConvT2d: stride 1, 64×64, 1 channel

To estimate the loss, our experiments estimate the full Jacobian matrix $J_g(z)$ using the finite difference method along each latent dimension. That is, for any given latent value z at which we wish to compute the Jacobian matrix, generate a set of k data points $z_i = z+\epsilon e_i$, where ϵ is a small fixed value and e_i a unit vector in the i^{th} latent direction. Then, run the forward model on the batch of z_i to generate $g(z_i)$ and estimate the i^{th} column of the Jacobian matrix as $(g(z_i) - g(z))/\epsilon$ This Jacobian matrix estimate is itself backward differentiable using standard backward differentiation, and can be directly used in our JL1-VAE loss.

A.13 JL1-VAE Three-dots Experiment Hyperparameters and Additional Results

For the three-dots dataset, we discretize the possible x,y coordinates of the center of each dot to 64 different values. The generative map is not injective, as the dots are identical, so the same resulting image can be formed from multiple permutations of ground-truth factor values. There are $64^6 \sim 68.7$ billion different possible input latent combinations, from which we pre-generate a cache of 500,000 images for training. During evaluation, new images are generated at runtime based on the desired ground-truth factors of variation.

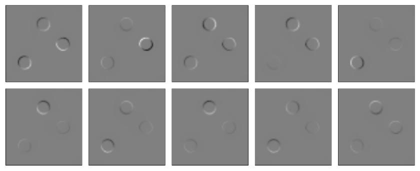
The beta-VAE was trained with $\beta = 4$ on a training dataset cache of 500,000 64×64-pixel images of three black dots on a white background, with x and y values for the dot centers appearing independently at one of 64 possible discrete locations, evenly spaced horizontally and vertically, across the image. A latent space of 10 dimensions was used. 300,000 independently sampled batches of 64 images from the cache were used, giving a total of 19,200,000 image presentations to the neural network. Additionally, the JL1-VAE was trained with the same β and model architecture on the same training dataset with our added L_1 regularization weighted by a hyperparameter $\gamma = 0.1$. The hyperparameter γ was chosen as the largest tested for which the learning algorithm converged to give good reconstruction accuracy. Linear annealing was used for both the β and γ parameters, annealing each from 0 to their final values over the first 100,000 batches. The Adam optimizer with a learning rate of 0.0001 was used

For the baseline comparison models, we use the implementations of β-VAE, FactorVAE, DIP-VAE-I, DIP-VAE-II, β-TCVAE, and AnnealedVAE from [55], matching their hyperparameter choices. For implementations that for which they provided a range of tested hyperparameters, we chose near the middle of their range. Thus, for β-VAE we used $\beta = 4$; for Annealed VAE we use $c_{max} = 25$, iteration threshold$= 100000$, and $\gamma = 1000$; for Factor VAE we use $\gamma = 30$; for DIP-VAE-I we use $\lambda_{od} = 5$ and $\lambda_d = 50$; for DIP-VAE-II we use $\lambda_{od} = 5$ and $\lambda_d = 5$; and for β-TCVAE we use $\beta = 4$.

The reference implementation provided with that work was modified in order to have consistent 4×4 kernels shown in the architecture in Table A.2. We include the modified implementation in our supplemental materials. The reference implementation of that architecture had unexplained 2×2 convolutional kernels for two layers.

We varied the random model initialization seed ten times and trained ten different models for each algorithm type. Additionally, we ran a smaller experiment randomizing both the model initialization seed and using different initial seeds for data sampling as well, getting similar results to those presented in the paper. The baseline implementation samples batches by epoch, shuffling after each epoch, while our implementation pulls independent random batches at each training set. Thus, the results for β-VAE in Figure 4.5 use independently-sampled random batches, while the results for β-VAE in 4.6 use shuffling after each epoch. This did not seem to affect results.

For the local metric calculations, we use the implementation provided by [55]. We sample 20 different local regions, pulling 10,000 points for each. We use a histogram discretization with 5 bins for mutual information calculations.


(a) Jacobian matrix columns for β-VAE

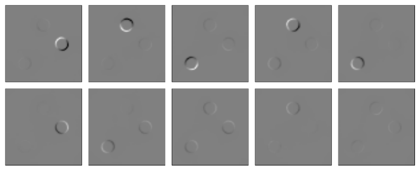
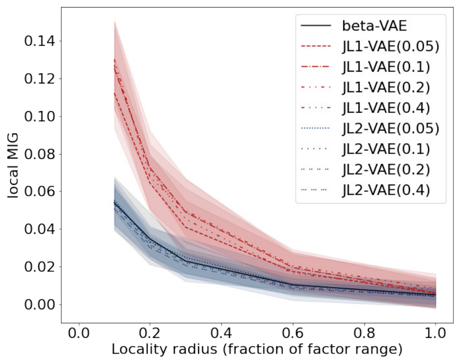

(b) Jacobian matrix columns for JL1-VAE

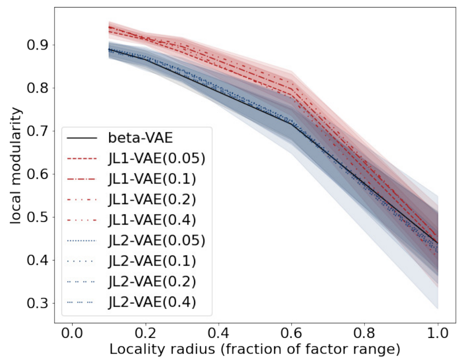
Figure A.7: Qualitative results for three-dots. Both models used $\beta = 4.0$. For JL1-VAE, $\gamma = 0.1$. All Jacobian matrix columns are shown.

All ten Jacobian columns associated with Figure 4.3 are shown in Figure A.7.

Additionally, we validate that an L_2 loss does not have the same disentangling properties as our L_1 loss by replacing the L_1 loss with an L_2 loss and computing the local disentanglement metrics in Figure A.8. We label the JL1-VAE with L_1 replaced by L_2 a JL2-VAE. All the L_2 regularizations are roughly indistinguishable from the β-VAE result, while the JL1-VAE consistently outperforms for the full range of tested regularization values γ.



(a) Local MIG scores



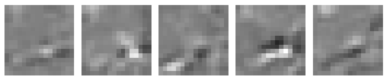
(b) Local modularity scores

Figure A.8: Local disentanglement scores varying the locality parameter ρ and the regularization factor γ for JL1-VAE, JL2-VAE, and β-VAE. The regularization factor γ is given in parentheses in the legend. Ten of each type of model were trained on the three-dots dataset with $\beta = 4$. This figure best viewed in color.

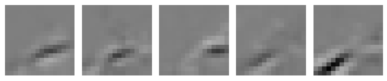
A.14 JL1-VAE Natural Image Experiment Hyperparameters and Additional Results

The beta-VAE was trained with $\beta = 0.01$ on a dataset 100,000 16x16-pixel crops from grayscale natural scenes [68], embedding the dataset into a latent space of 10 variables. We train for 100,000 batches of 128 images, re-shuffling the images after each epoch. Due to epoch endings, a few of the batches were incomplete, with fewer than 128 images. We train the JL1-VAE with the same model architecture and β on the same training dataset with our added L_1 regularization cost weighted by a hyperparameter $\gamma = 0.01$. The β was chosen as large as possible that still avoided significant dimensionality collapse, and then the hyperparameter γ was chosen as the largest tested for which the learning algorithm converged to give good reconstruction accuracy. Linear annealing was used for both the β and γ parameters, annealing each from 0 to their final values over the first 50,000 batches. The Adam optimizer was used with a learning rate of 0.001.

We show additional Jacobian column results, training on five latent dimensions in Figure A.9, and training on 25 latent dimensions in Figure A.10. Figure A.11 shows the top 100 PCA components and 100 trained ICA components.



(a) Jacobian columns for a β-VAE with 5 latent dimensions



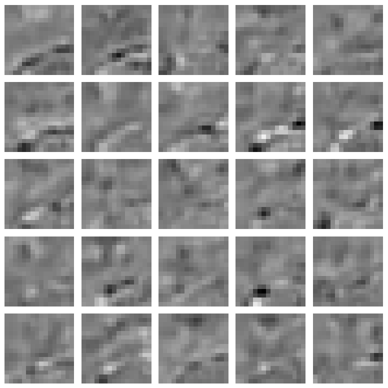
(b) Jacobian columns for a JL1-VAE with 5 latent dimensions

Figure A.9: Results for β-VAE and JL1-VAE using 5 latent dimensions (instead of the 10 shown in the main paper). Both are trained with $\beta = 0.01$, and JL1-VAE trained with $\gamma = 0.01$.

A.15 JL1-VAE MPI3D-Multi Experiment Hyperparameters

The beta-VAE was trained with $\beta = 0.01$ on a 2×2 tiling of every-other-pixel downsampling of randomly sampled images pulled from MPI3D-real, resulting in 64×64-pixel training images. We only sample MPI3D-real images of a top-down view of the robot holding a large, blue cube, with salmon background lighting. In this way, our dataset does not vary along unordered/sparse latent factors like color/shape. This leaves two independent dimensions of variance (horizontal and vertical axis joints) for each of the 4 tiled robot images. If we were to apply our local disentanglement metrics to discrete factors of variation that do not come with a natural distance metric such as "shape" or "color," we would require equality in order for values to be considered "close." That is, for any such factor of variation, a "local" sampled dataset will be constant on that factor.

The dataset was embedded into a latent space of 10 dimensions. 300,000 independently sampled batches of 64 images were taken from the cache, giving a total of 19,200,000 image presentations to the neural network. The β was chosen to give good reconsruction accuracy. Additionally, we train our JL1-VAE with


(a) Jacobian columns for a β-VAE with 25 latent dimensions

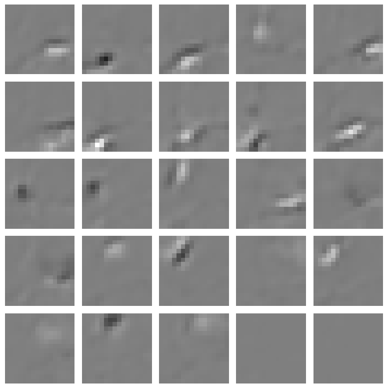
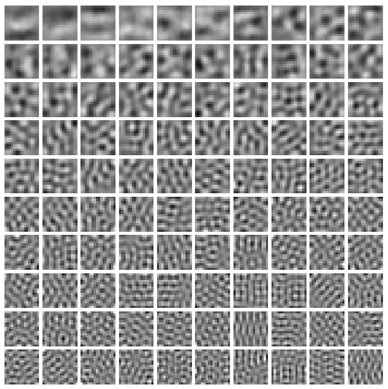
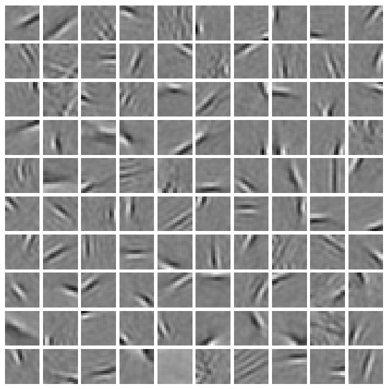

(b) Jacobian columns for a JL1-VAE with 25 latent dimensions

Figure A.10: Results for β-VAE and JL1-VAE using 25 latent dimensions (instead of the 10 shown in the main paper). Both are trained with $\beta = 0.01$, and JL1-VAE trained with $\gamma = 0.01$.

the same β and model architecture on the same training dataset with our added L_1 regularization weighted by a hyperparameter $\gamma = 0.01$. The hyperparameter γ was chosen as the largest tested for which the learning algorithm converged to give good reconstruction accuracy. Linear annealing was used for both the β and γ parameters, annealing each from 0 to their final values over the first 100,000 batches. The Adam optimizer was used with a learning rate of 0.0001.



(a) 100 latent PCA directions with the largest explained variance [70]



(b) 100 latent ICA directions fit using FastICA[70, 40]

Figure A.11: Latent vectors for PCA and ICA trained on random 16x16 crops from natural images collected by [68]

BIBLIOGRAPHY

[1] Samuel K. Ainsworth, Nicholas J. Foti, Adrian K. C. Lee, and Emily B. Fox. oi-VAE: Output interpretable VAEs for nonlinear group factor analysis. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 119–128. PMLR, 2018.

[2] Alexander A. Alemi, Ian Fischer, Joshua V. Dillon, and Kevin Murphy. Deep variational information bottleneck. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[3] Alexander A. Alemi, Ben Poole, Ian Fischer, Joshua V. Dillon, Rif A. Saurous, and Kevin Murphy. Fixing a broken ELBO. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 159–168. PMLR, 2018.

[4] Christopher G. Atkeson and Stefan Schaal. Robot learning from demonstration. In Douglas H. Fisher, editor, *Proceedings of the 14th International Conference on Machine Learning, ICML 1997, Nashville, Tennessee, USA, July 8-12, 1997*, pages 12–20. Morgan Kaufmann, 1997.

[5] Matthias Bauer and Andriy Mnih. Resampled priors for variational autoencoders. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*, volume 89 of *Proceedings of Machine Learning Research*, pages 66–75. PMLR, 2019.

[6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[7] Tapomayukh Bhattacharjee, Ethan K. Gordon, Rosario Scalise, Maria E. Cabrera, Anat Caspi, Maya Cakmak, and Siddhartha S. Srinivasa. Is more autonomy always better?: Exploring preferences of users with mobility impairments in robot-assisted feeding. In Tony Belpaeme, James E.

Young, Hatice Gunes, and Laurel D. Riek, editors, *HRI '20: ACM/IEEE International Conference on Human-Robot Interaction, Cambridge, United Kingdom, March 23-26, 2020*, pages 181–190. ACM, 2020.

[8] Tapomayukh Bhattacharjee, Hanjun Song, Gilwoo Lee, and Siddhartha S. Srinivasa. A dataset of food manipulation strategies. *https://doi.org/10.7910/DVN/8TTXZ7*, 2018.

[9] Christopher P. Burgess, Irina Higgins, Arka Pal, Loïc Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae. *CoRR*, abs/1804.03599, 2018.

[10] Xiaobin Chang, Frederick Tung, and Greg Mori. Learning discriminative prototypes with dynamic time warping. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 8395–8404. Computer Vision Foundation / IEEE, June 2021.

[11] Dong Chen and Hans-Georg Müller. Nonlinear manifold representations for functional data. *The Annals of Statistics*, 40(1):1 – 29, February 2012.

[12] Mingyu Chen, Ghassan AlRegib, and Biing-Hwang Juang. 6DMG: a new 6D motion gesture database. In *Proceedings of the 3rd Annual ACM SIGMM Conference on Multimedia Systems, MMSys 2012, Chapel Hill, NC, USA, February 22-24, 2012*, MMSys '12, page 83–88, New York, NY, USA, February 2012.

[13] Nutan Chen, Maximilian Karl, and Patrick van der Smagt. Dynamic movement primitives in latent space of time-dependent variational autoencoders. In *16th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2016, Cancun, Mexico, November 15-17, 2016*, pages 629–636. IEEE, November 2016.

[14] Nutan Chen, Alexej Klushyn, Francesco Ferroni, Justin Bayer, and Patrick Van Der Smagt. Learning flat latent manifolds with VAEs. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, July 2020.

[15] Ricky T. Q. Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. In Samy

Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, volume 31, pages 2615–2625. Curran Associates, Inc., December 2018.

[16] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, NeurIPS 2016, December 5-10, 2016, Barcelona, Spain*, volume 29, pages 2172–2180. Curran Associates, Inc., December 2016.

[17] Xinyu Chen, Jiajie Xu, Rui Zhou, Wei Chen, Junhua Fang, and Chengfei Liu. TrajVAE: A variational autoencoder model for trajectory generation. *Neurocomputing*, 428:332–339, 2021.

[18] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu, editors, *Robotics: Science and Systems XIX, RSS 2023, Daegu, Republic of Korea, July 10-14, 2023*, 2023.

[19] Cheol Jun Cho, Edward F. Chang, and Gopala Krishna Anumanchipalli. Neural latent aligner: Cross-trial alignment for learning representations of complex, naturalistic neural data. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 5661–5676. PMLR, 2023.

[20] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, May 2016.

[21] Marco Cuturi and Mathieu Blondel. Soft-DTW: a differentiable loss function for time-series. In Doina Precup and Yee Whye Teh, editors, *Proceed-*

ings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, volume 70 of *Proceedings of Machine Learning Research*, pages 894–903. PMLR, August 2017.

[22] Bruno Castro da Silva, George Dimitri Konidaris, and Andrew G. Barto. Learning parameterized skills. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

[23] Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical variational auto-encoders. In Amir Globerson and Ricardo Silva, editors, *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 856–865. AUAI Press, August 2018.

[24] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013, NeurIPS 2023, December 5-8, 2013, Lake Tahoe, Nevada, USA*, pages 2148–2156, December 2013.

[25] Ryan Feng, Youngsun Kim, Gilwoo Lee, Ethan K. Gordon, Matt Schmittle, Shivaum Kumar, Tapomayukh Bhattacharjee, and Siddhartha S. Srinivasa. Robot-assisted feeding: Generalizing skewering strategies across food items on a realistic plate. *CoRR*, abs/1906.02350, 2019.

[26] Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 22:1–72, 1906.

[27] Toni Giorgino. Computing and visualizing dynamic time warping alignments in R: The dtw package. *Journal of Statistical Software*, 31(7):1–24, 2009.

[28] Muhammad Waleed Gondal, Manuel Wuthrich, Djordje Miladinovic, Francesco Locatello, Martin Breidt, Valentin Volchkov, Joel Akpo, Olivier Bachem, Bernhard Schölkopf, and Stefan Bauer. On the transfer of inductive bias from simulation to the real world: a new disentanglement dataset. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in*

Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, volume 32, pages 15714–15725, 2019.

[29] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, NeurIPS 2014, December 8-13 2014, Montreal, Quebec, Canada*, volume 27, pages 2672–2680, December 2014.

[30] Alex Graves, Santiago Fernández, Faustino J. Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In William W. Cohen and Andrew W. Moore, editors, *Proceedings of the 23rd International Conference on Machine Learning, ICML 2006, Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 369–376. ACM, 2006.

[31] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Chris Burgess, Daniel Zoran, Loic Matthey, Matthew M. Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2424–2433. PMLR, June 2019.

[32] David Ha and Douglas Eck. A neural representation of sketch drawings. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[33] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2455–2467, 2018.

[34] Charles R. Hargraves and Stephen W. Paris. Direct trajectory optimiza-

tion using nonlinear programming and collocation. *Journal of Guidance, Control, and Dynamics*, 10(4):338–342, July 1987.

[35] Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[37] Guy Hoffman and Xuan Zhao. A primer for conducting experiments in human–robot interaction. *ACM Transactions on Human-Robot Interaction*, 10(1):1–31, October 2020.

[38] Judy Hoffman, Daniel A. Roberts, and Sho Yaida. Robust learning with Jacobian regularization. *CoRR*, abs/1908.02729, 2019.

[39] Rachel M. Holladay, Oren Salzman, and Siddhartha S. Srinivasa. Minimizing task-space fréchet error via efficient incremental graph search. *IEEE Robotics and Automation Letters*, 4(2):1999–2006, 2019.

[40] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

[41] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, 2013.

[42] Shervin Javdani, Henny Admoni, Stefania Pellegrinelli, Siddhartha S. Srinivasa, and J. Andrew Bagnell. Shared autonomy via hindsight optimization for teleoperation and teaming. *The International Journal of Robotics Research*, 37(7):717–742, 2018.

[43] Linxing Preston Jiang and Luciano de la Iglesia. Improved training of sparse coding variational autoencoder via weight normalization. *CoRR*, abs/2101.09453, 2021.

[44] Ian T. Jolliffe, Nickolay T. Trendafilov, and Mudassir Uddin. A modified principal component technique based on the LASSO. *Journal of Computational and Graphical Statistics*, 12(3):531–547, September 2003.

[45] Rayyan Ahmad Khan, Muhammad Umer Anwaar, and Martin Kleinsteuber. Epitomic variational graph autoencoder. In *25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021*, pages 7203–7210. IEEE, 2020.

[46] Ilyes Khemakhem, Diederik P. Kingma, Ricardo Pio Monti, and Aapo Hyvärinen. Variational autoencoders and nonlinear ICA: A unifying framework. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 2207–2217. PMLR, 2020.

[47] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2654–2663. PMLR, 2018.

[48] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[49] Kinova. Gen3 robots. *https://www.kinovarobotics.com/product/gen3-robots*, 2021.

[50] Alois Kneip and James O. Ramsay. Combining registration and fitting for functional models. *Journal of the American Statistical Association*, 103(483):1155–1165, 2008.

[51] Kaushik Koneripalli, Suhas Lohit, Rushil Anirudh, and Pavan K. Turaga. Rate-invariant autoencoding of time-series. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pages 3732–3736. IEEE, 2020.

[52] Joseph B. Kruskal and Mark Liberman. The symmetric time-warping problem: From continuous to discrete. In David Sankoff and Joseph B.

Kruskal, editors, *Time Warps, String Edits, and Macromolecules : The Theory and Practice of Sequence Comparison*, chapter 4, pages 125–161. Addison-Wesley Publishing Company, Advanced Book Program, Reading, Mass., 1983.

[53] Jannick Kuester, Wolfgang Gross, and Wolfgang Middelmann. 1D-convolutional autoencoder based hyperspectral data compression. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B1-2021:15–21, 2021.

[54] Abhishek Kumar and Ben Poole. On implicit regularization in β-VAEs. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5480–5490. PMLR, 2020.

[55] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Rätsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 4114–4124. PMLR, 2019.

[56] Francesco Locatello, Ben Poole, Gunnar Rätsch, Bernhard Schölkopf, Olivier Bachem, and Michael Tschannen. Weakly-supervised disentanglement without compromises. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 6348–6359. PMLR, 2020.

[57] Dylan P. Losey, Krishnan Srinivasan, Ajay Mandlekar, Animesh Garg, and Dorsa Sadigh. Controlling assistive robots with learned latent actions. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, pages 378–384. IEEE, 2020.

[58] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through L_0 regularization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[59] Xiaoyu Lu, Jan Stuehmer, and Katja Hofmann. Trajectory VAE for multi-modal imitation. *https://openreview.net/forum?id=Byx1VnR9K7*, 2019.

[60] Alireza Makhzani and Brendan J. Frey. k-Sparse autoencoders. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[61] Emile Mathieu, Tom Rainforth, N. Siddharth, and Yee Whye Teh. Disentangling disentanglement in variational autoencoders. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 4402–4412. PMLR, 2019.

[62] Takamitsu Matsubara, Sang-Ho Hyon, and Jun Morimoto. Learning parametric dynamic movement primitives from multiple demonstrations. *Neural Networks*, 24(5):493–500, 2011.

[63] Hiroyuki Miyamoto, Francesca Gandolfo, Hiroaki Gomi, Stefan Schaal, Yasuharu Koike, Rieko Osu, Eri Nakano, Yasuhiro Wada, and Mitsuo Kawato. A kendama learning robot based on a dynamic optimization theory. In *Proceedings 4th IEEE International Workshop on Robot and Human Communication, ROMAN 1995, 5-7 July 1995, Tokyo, Japan*, ROMAN-95. IEEE, July 1995.

[64] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. Robustness via curvature regularization, and vice versa. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 9078–9086. Computer Vision Foundation / IEEE, 2019.

[65] Andrew Ng. CS294A lecture notes sparse autoencoder. *http://www.stanford.edu/class/cs294a/*, pages 1–19, 2011.

[66] NVIDIA. Physx. *https://developer.nvidia.com/physx-sdk*, 2023.

[67] Bruno A Olshausen. Learning linear, sparse, factorial codes. Technical Report A.I. Memo No. 1580, MIT, September 1996.

[68] Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, June 1996.

[69] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, November 1901.

[70] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[71] Luis A. Pérez Rey, Vlado Menkovski, and Jim Portegies. Diffusion variational autoencoders. In Christian Bessiere, editor, *Proceedings of the 29th International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 2704–2710. ijcai.org, 2020.

[72] Affan Pervez and Dongheui Lee. Learning task-parameterized dynamic movement primitives using mixture of GMMs. *Intelligent Service Robotics*, 11(1):61–78, 2018.

[73] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

[74] François Petitjean, Alain Ketterlin, and Pierre Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44(3):678–693, 2011.

[75] Travers Rhodes, Tapomayukh Bhattacharjee, and Daniel D. Lee. Learning from demonstration using a curvature regularized variational autoencoder (CurvVAE). In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2022, Kyoto, Japan, October 23-27, 2022*, pages 10795–10800. IEEE, 2022.

[76] Travers Rhodes and Daniel D. Lee. Local disentanglement in variational auto-encoders using jacobian l_1 regularization. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman

Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 22708–22719, 2021.

[77] Travers Rhodes and Daniel D. Lee. TimewarpVAE: Simultaneous time-warping and representation learning of trajectories. *CoRR*, abs/2310.16027, 2024.

[78] Karl Ridgeway and Michael C. Mozer. Learning deep disentangled embeddings with the F-statistic loss. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 185–194, 2018.

[79] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 833–840. Omnipress, 2011.

[80] Robotiq. 2f-85 gripper. *https://robotiq.com/products/2f85-140-adaptive-robot-gripper*, 2021.

[81] Michal Rolínek, Dominik Zietlow, and Georg Martius. Variational autoencoders pursue PCA directions (by accident). In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 12406–12415. Computer Vision Foundation / IEEE, 2019.

[82] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, February 1978.

[83] Stefan Schaal, Peyman Mohajerian, and Auke Ijspeert. Dynamics systems vs. optimal control — a unifying view. In Paul Cisek, Trevor Drew, and John F. Kalaska, editors, *Computational Neuroscience: Theoretical Insights into Brain Function*, volume 165 of *Progress in Brain Research*, pages 425–445. Elsevier, 2007.

[84] Mona Schirmer, Mazin Eltayeb, Stefan Lessmann, and Maja Rudolph.

Modeling irregular time series with continuous recurrent units. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 19388–19405. PMLR, 2022.

[85] David Schultz and Brijnesh J. Jain. Nonsmooth analysis and subgradient methods for averaging in dynamic time warping spaces. *Pattern Recognition*, 74:340–358, 2018.

[86] H. Sebastian Seung and Daniel D. Lee. The manifold ways of perception. *Science*, 290(5500):2268–2269, December 2000.

[87] Jan Stuehmer, Richard E. Turner, and Sebastian Nowozin. Independent subspace analysis for unsupervised learning of disentangled representations. In Silvia Chiappa and Roberto Calandra, editors, *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, volume 108 of *Proceedings of Machine Learning Research*, pages 1200–1210. PMLR, 2020.

[88] Dídac Surís Coll-Vinent and Carl Vondrick. Representing spatial trajectories as distributions. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, December 2022.

[89] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics. *https://drake.mit.edu*, 2019.

[90] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[91] Jakub M. Tomczak and Max Welling. VAE with a VampPrior. In Amos J. Storkey and Fernando Pérez-Cruz, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, volume 84 of *Proceedings of Machine Learning Research*, pages 1214–1223. PMLR, 2018.

[92] George Trigeorgis, Mihalis A. Nicolaou, Björn W. Schuller, and Stefanos

Zafeiriou. Deep canonical time warping for simultaneous alignment and representation learning of sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1128–1138, 2018.

[93] Dániel Varga, Adrián Csiszárik, and Zsolt Zombori. Gradient regularization improves accuracy of discriminative models. *Schedae Informaticae*, 27:31–45, 2018.

[94] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, NeurIPS 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[95] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[96] Jane-Ling Wang, Jeng-Min Chiou, and Hans-Georg Müller. Functional data analysis. *Annual Review of Statistics and Its Application*, 3(1):257–295, June 2016.

[97] Nicholas Watters, Loïc Matthey, Christopher P. Burgess, and Alexander Lerchner. Spatial broadcast decoder: A simple architecture for learning disentangled representations in VAEs. *CoRR*, abs/1901.07017, 2019.

[98] Ron Shapira Weber, Matan Eyal, Nicki Skafte Detlefsen, Oren Shriki, and Oren Freifeld. Diffeomorphic temporal alignment nets. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 6570–6581, 2019.

[99] Alex H. Williams, Ben Poole, Niru Maheswaranathan, Ashesh K. Dhawale, Tucker Fisher, Christopher D. Wilson, David H. Brann, Eric M. Trautmann, Stephen Ryu, Roman Shusterman, Dmitry Rinberg, Bence P. Ölveczky, Krishna V. Shenoy, and Surya Ganguli. Discovering precise

temporal patterns in large-scale neural recordings through robust and interpretable time warping. *Neuron*, 105(2):246–259.e8, January 2020.

[100] Shengjia Zhao, Hongyu Ren, Arianna Yuan, Jiaming Song, Noah D. Goodman, and Stefano Ermon. Bias and generalization in deep generative models: An empirical study. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 10815–10824, 2018.

[101] Feng Zhou and Fernando De la Torre. Generalized time warping for multi-modal alignment of human motion. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2012, Providence, RI, USA, June 16-21, 2012*, pages 1282–1289. IEEE Computer Society, 2012.

[102] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong T. Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of the 7th Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR, 2023.

[103] Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, June 2006.